ADMIN:DOCS

Windows Server 2025 Core: Installation and Administration via SSH and PowerShell

Operate Windows Server 2025 Core on Proxmox VE 9 without a desktop GUI: From VM creation and initial SConfig setup to OpenSSH and headless PowerShell management.

Category: Windows Server **Published:** 10.11.2026

Running Windows Server with a full desktop interface (Desktop Experience) on a virtualization host such as Proxmox VE 9 locks up considerable host resources: several gigabytes of unneeded physical memory, continuous CPU baseline consumption for graphics compositors and window management subsystems, and an unnecessarily large virtual disk footprint filled with UI libraries that remain completely untouched during headless production operations. Furthermore, the graphical stack significantly widens the system attack surface, regularly forcing extensive monthly cumulative update installation cycles and disruptive host-level reboots.

Windows Server 2025 Core strips out the Windows Explorer shell, legacy Control Panel applets, and all local graphical management tools. The operating system boots directly into a lean, text-based console. For heterogeneous environments, this architectural model aligns Windows infrastructure with established Linux operational paradigms: instances run completely headless, minimize resource footprint, and rely on standardized, scriptable network protocols for remote lifecycle management.

In combination with Proxmox VE 9 and Linux-based administrative workstations, OpenSSH for Windows under Windows Server 2025 bridges the traditional operational gap between both platforms. The OpenSSH server component is natively integrated into Windows Server 2025 OS media, eliminating the fragile WinRM HTTPS certificate configurations, Kerberos double-hop workarounds, and cross-platform transport layers previously required to reach a remote Windows shell from Linux. Remote access occurs straight from standard Linux terminals through

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

authenticated OpenSSH key pairs directly into a native Windows PowerShell session.

The objective of this guide is building a lean, auditable, and resilient Server Core instance on Proxmox VE 9

Initial network and baseline configuration is performed via SConfig, hardened using persistent OpenSSH daemon key authentication, and subsequently operated exclusively through PowerShell cmdlets and native console utilities over the wire.

> **Prerequisites and Lab Environment:** A reproducible deployment of Windows Server Core requires a clear understanding of the target audience, the underlying hypervisor hardware, and the practical distinction between minimum vendor specifications and viable lab resources.

Target Audience and Required Background

This guide is designed for system administrators who are already familiar with standard operational infrastructure tasks:

- Solid grasp of core operating system concepts, processes, user accounts, and service privilege boundaries
- Practical experience managing IP networks, subnets, routing logic, and domain name resolution
- Working familiarity with virtualization platforms such as Proxmox VE or KVM-based Hypervisor and VM Management
- Confident baseline navigation on an administrative command line

Hands-on experience with Linux environments and standard OpenSSH clients makes remote terminal management intuitive, though prior Windows Server Core or PowerShell expertise is not strictly required. Every administrative workflow, cmdlet, and registry adjustment is systematically introduced, contextualized, and verified as we proceed.

System Requirements and Lab Recommendations

Microsoft outlines official minimum hardware requirements for Windows Server 2025 that suffice to initialize the NT kernel, but fall well short of providing a stable virtualization baseline for real workloads:

Official Minimum Requirement	
Processor	1.4 GHz x64 (x86-64-v2/v3)
System Memory	2 GiB RAM (Server Core)
Primary Storage	32 GiB hard drive
Firmware & Machine	UEFI 2.3.1c (Secure Boot)
Network Adapter	1 Gbit/s Ethernet
Guest Integration	Not specified

Recommended Lab Configuration (PVE)

Processor
: 2 to 4 vCPUs (Type host)

System Memory
: 4 to 8 GiB RAM

Primary Storage
: 64 GiB (VirtIO-SCSI)

Firmware & Machine
: OVMF (UEFI), Machine Type Q35

Network Adapter
: VirtIO (paravirtualized)

Guest Integration
: QEMU Guest Agent (qemu-ga)

Common: 64-bit x86-64 architecture, fully functional as a KVM guest on Proxmox VE 9

The selection of virtual storage controllers and network adapters directly dictates the installation workflow. While paravirtualized VirtIO devices deliver the lowest hypervisor overhead and highest sustained throughput, default Windows Server installation media lacks out-of-the-box VirtIO drivers. These must either be loaded interactively during the setup stage from the `virtio-win` driver ISO or bypassed initially using emulated hardware.

Documented Test Configuration

All operational steps, configuration files, and command outputs documented throughout this guide were executed and validated on a live Proxmox VE 9 cluster node. To deliberately minimize driver dependencies during initial unattended and interactive setup stages, the baseline deployment uses emulated SATA storage and Intel network adapters:

Component	Configuration Specification
Hypervisor	Proxmox VE 9
Guest Operating System	Windows Server 2025 Standard Evaluation (x64)

Component	Configuration Specification
Installation Variant	Server Core (headless, without Desktop Experience)
Machine Type & Firmware	Q35, OVMF (UEFI) with 4M EFI Disk
Processor Allocation	6 vCPU, CPU Type `host`
Assigned Memory	14,096 MiB RAM
System Drive	150 GiB virtual disk, Interface: SATA
Network Interface	Intel E1000
Driver ISO	virtio-win-0.1.302 (attached for post-install guest tooling)
Management Client	Linux Workstation with native OpenSSH client

> **Hardware Emulation vs. Paravirtualization:** Using SATA and Intel E1000 enables an uninterrupted installation workflow because Windows Server 2025 includes inbox drivers for both controller types. In high-density production environments on Proxmox VE, paravirtualized VirtIO-SCSI and VirtIO-Net remain the architectural best practice to avoid QEMU emulation overhead. Manual driver injection is addressed transparently in the installation section.

The assigned 14 GiB of RAM and 6 vCPUs reflect the available host capacity of our test bench and do not represent hard minimums. Specialized host-level overrides (such as custom SMBIOS strings or strict CPU core pinning) are unnecessary for regular operations and have been omitted intentionally.

Network Topology and Port Matrix for Administration

In heterogeneous infrastructures where Linux workstations, Proxmox clusters, and Windows servers are isolated across distinct VLANs and firewalls, administrative ports must be strictly defined. Because Server Core eliminates browser-based local configuration applets, all network traffic relies on standardized services:

Port	Protocol	Service / Purpose	Operational Requirement
22	TCP	OpenSSH Server (sshd)	Primary remote terminal administration from Linux and Ansible
53	UDP/TCP	Domain Name System (DNS)	Local network name resolution and Active Directory SRV queries
88	UDP/TCP	Kerberos v5	Authentication tickets during Active Directory domain operations
123	UDP	Network Time Protocol (W32Time)	Time synchronization to prevent Kerberos clock drift failures
135	TCP	RPC Endpoint Mapper	Mandatory for remote MMC snap-ins, RSAT, and legacy WMI calls
389 / 636	TCP	LDAP / LDAPS	Active Directory directory queries and object lookups
445	TCP	Server Message Block (SMB)	Remote file transfers, script distribution, and named pipes
3389	TCP/UDP	Remote Desktop Protocol (RDP)	Optional text console terminal sessions via RDP clients
5985	TCP	WinRM HTTP	PowerShell Remoting (message-level encryption via Kerberos/NTLM; transport without TLS)
5986	TCP	WinRM HTTPS	PowerShell Remoting with TLS transport encryption and server certificate
8006	TCP	Proxmox VE Web GUI	Hypervisor management and noVNC console access

Detailed Storage Sizing and Partitioning Strategy

Compared to a full Desktop Experience installation (which claims between 24 and 28 GiB immediately upon completion), a fresh Windows Server 2025 Core instance exhibits a remarkably compact disk footprint:

- **Core Operating System (`C:\Windows`):** approximately 11 to 13 GiB.
- **Component Store (`C:\Windows\WinSxS`):** approximately 5 to 7 GiB for hardlinked system components, service packs, and update staging.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

- **Virtual Memory (`pagefile.sys`):** dynamically managed by default (typically allocating 2 to 4 GiB under 8 GiB physical RAM).
- **Kernel Crash Dump Reservation (`MEMORY.DMP`):** configured up to full physical RAM capacity for comprehensive crash diagnostics.
- **Staging Buffer for Monthly Quality Updates:** cumulative update packages unpack temporary CAB/MSU staging archives ranging between 4 and 8 GiB during installation.

For an isolated system drive on Proxmox VE, a minimum virtual disk allocation of **60 to 80 GiB** is strongly advised. This headroom accommodates multiple years of cumulative updates, driver rollbacks, and log retention without requiring emergency administrative disk expansion.

Understanding Windows Server Core

Windows Server Core is not a separately licensed SKU or stripped-down edition, but an official installation option available in both Windows Server 2025 Standard and Datacenter editions. The operating system omits the traditional Explorer shell: there is no desktop, no taskbar, no notification area, and no local Start menu. Upon initial boot, the kernel exclusively initializes the console subsystem and spawns a single command prompt window.

> **Headless Operational Discipline:** On Server Core, you interact solely with a console window upon logon. If you accidentally close this window, the screen remains entirely blank. Pressing `Ctrl` + `Shift` + `Esc` (or sending `Ctrl` + `Alt` + `Del` via the Proxmox noVNC menu) launches Task Manager, allowing you to spawn a fresh shell via *File → Run new task* by entering `powershell.exe` or `cmd.exe`.

Architecture and Differences from Desktop Experience

In traditional Windows Server installations with Desktop Experience, the graphical stack is tightly coupled with core operating system libraries. Subsystems handling window composition, GDI font rendering, visual themes, and local shell hooks reside permanently on disk and load into physical memory at boot.

Server Core strips these user-facing subsystems out almost entirely. Binary executables for `explorer.exe`, `mmc.exe`, and associated local setup wizards simply do not exist in the filesystem. This structural reduction yields three concrete operational benefits:

- **Minimized Attack Surface:** A significant percentage of Windows security vulnerabilities historically target client-oriented desktop components—such as graphics rendering engines, HTML shell helpers, or audio stacks. On Server Core, these libraries are completely absent.
- **Lower Resource Footprint:** Eliminating desktop graphics conserves CPU cycles and physical RAM on the hypervisor host. Host compute resources remain dedicated to backend production services rather than rendering desktop widgets.
- **Reduced Update and Reboot Frequency:** Because cumulative monthly patches targeting GUI components do not apply to Server Core, fewer update packages require installation, directly translating to fewer mandatory service reboots.

Desktop Experience

Local User Interface
: Windows Explorer, Start Menu, Taskbar

Graphical Management
: Local MMC snap-ins and GUI wizards

Resource Overhead
: Higher baseline memory and disk I/O

Patch Maintenance
: Higher patch frequency for desktop packages

Attack Surface
: Broad attack surface across UI libraries

Server Core

Local User Interface
: Pure text console (cmd.exe / powershell.exe)

Graphical Management
: Exclusively handled through remote tooling

Resource Overhead
: Very low memory footprint and compact disk usage

Patch Maintenance
: Fewer update packages, significantly fewer reboots

Attack Surface
: Restrained to NT kernel, networking, and server roles

Common: Identical Windows Server 2025 NT kernel, identical driver model, and standard Win32/CIM APIs

Complete Role Compatibility Matrix for Windows Server 2025 Core

A crucial architectural consideration prior to deploying Server Core is verifying server role compatibility. Nearly all core datacenter infrastructure roles operate on Server Core without restriction. Specific roles requiring local end-user desktop sessions or legacy GUI frameworks are strictly unsupported:

Server Role	Support on Server Core	Feature Identifier (PowerShell)	Management Approach
Active Directory Domain Services	Fully Supported	`AD-Domain-Services`	PowerShell (`ActiveDirectory`), Remote RSAT
Active Directory Certificate Services	Fully Supported	`AD-Certificate`	PowerShell (`ADCSAdministration`), `certutil.exe`
Active Directory Federation Services	Fully Supported	`ADFS-Federation`	PowerShell (`ADFS`)

Server Role	Support on Server Core	Feature Identifier (PowerShell)	Management Approach
Active Directory Lightweight Directory Services	✅ Fully Supported	`ADLDS`	PowerShell, Remote RSAT
Active Directory Rights Management	✅ Fully Supported	`ADRMS`	PowerShell, Remote Web Console
DHCP Server	✅ Fully Supported	`DHCP`	PowerShell (`DhcpServer`), Remote RSAT
DNS Server	✅ Fully Supported	`DNS`	PowerShell (`DnsServer`), `dnscmd.exe`
File and Storage Services	✅ Fully Supported	`File-Services`, `FS-FileServer`	PowerShell (`Storage`, `SmbShare`)
Web Server (IIS)	✅ Fully Supported	`Web-Server`	PowerShell (`WebAdministration`), AppCmd
Hyper-V	✅ Fully Supported	`Hyper-V`	PowerShell (`Hyper-V`), Remote Hyper-V Manager
Print and Document Services	✅ Fully Supported	`Print-Server`	PowerShell (`PrintManagement`)
Windows Server Update Services (WSUS)	✅ Fully Supported	`UpdateServices`	PowerShell (`UpdateServices`), SQL/WID
Remote Desktop Services: Connection Broker	✅ Fully Supported	`RDS-Connection-Broker`	PowerShell (`RemoteDesktop`)
Remote Desktop Services: Gateway	✅ Fully Supported	`RDS-Gateway`	PowerShell, Remote RSAT
Remote Desktop Services: Web Access	✅ Fully Supported	`RDS-Web-Access`	PowerShell, Web Portal
Remote Desktop Services: Session Host	❌ Unsupported	Not Available	Strictly requires Desktop Experience
Network Policy and Access Services (NPS)	⚠ Constrained	`NPAS`	Exclusively via Remote RSAT (no local GUI)

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

Server Role	Support on Server Core	Feature Identifier (PowerShell)	Management Approach
Windows Deployment Services (WDS)	⚠ Constrained	`WDS`	`wdsutil.exe`, Remote RSAT
Fax Server	❌ Unsupported	Not Available	Deprecated legacy role; requires desktop stack

> ⚠️ **Permanent Architectural Choice:** Since Windows Server 2016, switching between Server Core and Desktop Experience after installation is no longer supported. The deployment choice is permanent. Transitioning an existing instance requires a complete reinstallation of the guest operating system.

Local and Remote Administration Tooling

Operating without a local desktop environment relies on a structured hierarchy of text-based configuration utilities, object pipelines, and remote protocols:

- **SConfig (`sconfig.cmd` / `sconfig.ps1`):** The built-in utility for Server Core initial setup is a menu-driven console application rewritten natively in PowerShell in modern Windows Server versions. It streamlines foundational configuration directly at the hypervisor console (such as Proxmox noVNC), allowing administrators to set hostnames, IP addressing, domain joins, and update rings without remembering lengthy commands.
- **Windows PowerShell:** The object-oriented shell environment is the primary tool for all operational administration. Rather than parsing raw unstructured text strings, PowerShell processes strongly typed .NET objects across the pipeline. Local system settings, services, firewall rules, and server roles are managed through standardized cmdlets.
- **WinRM (Windows Remote Management):** Microsoft's traditional remote management protocol implements the SOAP- and HTTP/HTTPS-based WS-Management standard (TCP ports 5985 and 5986). WinRM forms the foundation for PowerShell Remoting (`Enter-PSSession -ComputerName`), Windows Admin Center (WAC), and remote MMC snap-ins connecting from other Windows management machines.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

- **OpenSSH:** The open standard for encrypted terminal sessions over TCP port 22. OpenSSH encapsulates the connection within a hardened cryptographic tunnel and hands execution over to a native system shell (configured to PowerShell). It enables seamless remote access from Linux and macOS workstations without requiring Windows-specific authentication protocols like Kerberos or NTLM over HTTP.

> **Why SSH Excels in Heterogeneous Networks:** WinRM encrypts message payloads at the protocol level when authenticating via Kerberos or NTLM over HTTP (port 5985). Outside Active Directory domains, however, verifying server identity requires either deploying custom TLS certificates for HTTPS (port 5986) or configuring local `TrustedHosts` lists. `TrustedHosts` waives server authentication without providing independent encryption. OpenSSH relies on the established public-key cryptography and host-key verification standard familiar from Linux System Hardening.

The Boundary: SConfig Remote Management vs. OpenSSH Daemon

A frequent misconception during the initial setup of Server Core concerns Option 4 in the SConfig menu: *Configure Remote Management*.

Enabling this option exclusively configures firewall exceptions and local listeners for WinRM, along with legacy WMI and Remote Desktop exceptions (if RDP is explicitly activated).

> ⚠ **Common Operational Trap:** Enabling Remote Management in SConfig does not interact with the OpenSSH Server (`sshd`) in any way. The Windows Firewall does not open TCP port 22, nor is the SSH service configured for automated startup. Disconnecting from the console after configuring SConfig will leave the machine unreachable via SSH.

This architectural boundary dictates concrete operational practice:

- OpenSSH functions as an independent Windows system service under Windows Server 2025.

- Service startup type, runtime status, and inbound firewall rules for TCP port 22 must be configured explicitly through PowerShell.
- For administrators managing systems from Linux workstations, SConfig Option 4 is strictly optional, while dedicated PowerShell configuration of `sshd` is mandatory.

The Dual-Stack Management Architecture: OpenSSH and Windows Admin Center (WAC)

In modern enterprise environments, operating without a local desktop interface does not preclude graphical overview dashboards. Professional deployments adopt a pragmatic dual-stack management architecture:

1. **Terminal and Automation Layer (OpenSSH):**

Designed for rapid ad-hoc troubleshooting, direct shell scripting, and integration into cross-platform automation tools like Ansible, Terraform, and CI/CD deployment agents. Over port 22, administrators connect from Linux terminals directly into native PowerShell 5.1/7 sessions without RPC or Kerberos dependencies.

1. **Web-Based Management Layer (Windows Admin Center):**

Microsoft's browser-based management tool (Windows Admin Center, WAC) deploys on a dedicated gateway server or administrative workstation. WAC communicates over WinRM (port 5985) and PowerShell Remoting in the background, rendering a modern HTML5 interface in the browser:

- Real-time performance graphing for CPU load, memory pressure, and disk IOPS
- Certificate store auditing and firewall rule management
- Storage Spaces pool configuration and virtual disk allocation
- Searchable Windows Event Log viewer with structured export capabilities

This architecture combines the strengths of both approaches: the managed host remains lean, secure, and devoid of GUI bloat, while administrators choose flexibly between rapid terminal execution and visual web dashboards according to the task at hand.

Creating the Virtual Machine under Proxmox VE 9

With the architectural foundation established, we proceed to creating the virtual machine on the hypervisor. Rather than navigating the Proxmox web interface, we define the VM entirely through the host shell using the `qm` (QEMU/KVM Virtual Machine Manager) CLI utility. This ensures a fully reproducible, version-controlled hardware configuration.

The provisioning workflow proceeds through four sequential stages:

1. **Verify Storage & ISO Media** Confirm Windows Server and VirtIO ISO availability on Proxmox storage
2. **Initialize Base VM via qm** Define Q35 machine type, UEFI (OVMF) firmware, CPU cores, and memory
3. **Configure Disks & EFI Storage** Allocate EFI disk, configure primary storage, and set boot order
4. **Enable Guest Integration & Launch** Activate QEMU Guest Agent, attach vTPM, and open noVNC console

Preparing Installation Media

Before invoking `qm create`, verify that the required installation media reside on the target Proxmox storage pool: the official Windows Server 2025 installation ISO and the latest driver ISO aligned with VirtIO Best Practices. Inspect available images using the Proxmox storage manager `pvesm`:

```
pvesm list local --content iso
```

The command lists stored volume identifiers, such as `local:iso/Windows_Server_2025_Evaluation_x64.iso` and `local:iso/virtio-win-0.1.302.iso`. These exact strings are assigned to virtual optical drives.

Creating the VM via CLI Tool qm

Under Proxmox VE 9, `qm create` controls the complete allocation of KVM virtualization resources. For modern Windows guests, machine architecture, firmware type, and CPU features are decisive:

```
qm create 104 \
  --name "WIN-CORE01" \
  --ostype win11 \
  --machine q35 \
  --bios ovmf \
  --cpu host \
  --cores 4 \
  --sockets 1 \
  --memory 8192 \
  --agent enabled=1
```

> **Significance of OS Type `win11`:** Proxmox VE uses `--ostype win11` as the umbrella identifier for modern 64-bit Windows operating systems, including Windows 11 and Windows Server 2022/2025. This flag instructs QEMU to enable essential Hyper-V enlightenments (such as synic, stimer, and relaxed) that significantly reduce kernel virtualization overhead.

Detailed parameter breakdown:

- `--machine q35`: Emulates the modern PCIe-based Q35 chipset rather than the legacy PCI-only `i440fx` bus.

- `--bios ovmf`: Configures UEFI firmware instead of legacy SeaBIOS. Windows Server 2025 requires UEFI for modern GPT partition layouts and Secure Boot.
- `--cpu host`: Passes physical host CPU instructions directly through to the virtual machine.

> ⚠ **Performance Trap of `kvm64`:** Leaving the CPU type at default `kvm64` emulates an outdated generic processor lacking modern instruction sets like AVX, AVX2, AES-NI, and SSE4.2. On Windows Server 2025, this causes severe latency during cryptographic operations (such as SSH handshakes and TLS handshakes) and can trigger setup errors.

- `--agent enabled=1`: Signals Proxmox VE to communicate with the QEMU Guest Agent once installed, enabling accurate IP reporting in the web GUI and consistent snapshot backups via Proxmox Backup Server (PBS).

Allocating EFI Disk and Virtual TPM

Because UEFI (`ovmf`) was specified as the firmware, the virtual machine requires a dedicated EFI storage volume to store non-volatile NVRAM variables, including boot entries and Secure Boot signature databases:

```
qm set 104 --efidisk0 local-zfs:1,efitype=4m,pre-enrolled-keys=1
```

The `pre-enrolled-keys=1` flag injects standard Microsoft Secure Boot certificates directly into the virtual NVRAM store.

To evaluate BitLocker drive encryption or Virtualization-based Security (VBS) in later hardening phases, attach a virtualized TPM 2.0 state disk:

```
qm set 104 --tpmstate0 local-zfs:1,version=v2.0
```

Configuring Storage Controllers and Virtual Disks

At this point, choose your virtual storage controller architecture:

- **Variant A: VirtIO-SCSI (Production Performance Best Practice)**

Paravirtualized controller delivering minimal CPU overhead and maximum throughput. Requires manual driver loading during OS installation:

```
qm set 104 \
  --scsihw virtio-scsi-single \
  --scsi0 local-zfs:64,discard=on,ssd=1
```

`discard=on` forwards TRIM commands back to the underlying ZFS pool, and `ssd=1` flags the virtual drive as solid-state media within Windows.

- **Variant B: SATA (Direct Setup without Custom Drivers)**

Reflects our documented test lab configuration. Windows Server 2025 recognizes the disk out-of-the-box:

```
qm set 104 --sata0 local-zfs:150
```

Network Adapters, Optical Media, and Boot Order

In the final hardware definition step, link the network interface, attach ISO images to virtual CD/DVD drives, and establish the primary boot device:

- **Network:** While VirtIO (`--net0 virtio,bridge=vmbr0`) is standard for production, our documented test setup uses Intel E1000 emulation to bypass setup-time network driver dependencies:

```
qm set 104 --net0 e1000,bridge=vmbr0
```

- **Attach Installation Media:**

```
qm set 104 \
  --ide2 local:iso/Windows_Server_2025_Evaluation_x64.iso,media=cdrom \
  --ide0 local:iso/virtio-win-0.1.302.iso,media=cdrom
```

> **Keep VirtIO ISO Permanently Attached:** Even when selecting SATA and Intel E1000 for initial setup, keep `virtio-win.iso` connected to `ide0`. This ensures guest agent installers (`qemu-ga-x86_64.msi`) and VirtIO drivers remain accessible immediately after booting.

- **Configure Boot Order to prioritize the Windows setup ISO:**

```
qm set 104 --boot order=ide2
```

Full Reference Configuration

To deploy the entire virtual machine in a single reproducible command, run the combined CLI invocation below. Adjust the VM ID (`104`), storage pool (`local-zfs`), and exact ISO names to match your cluster environment:

```
qm create 104 \
  --name "WIN-CORE01" \
  --ostype win11 \
  --machine q35 \
  --bios ovmf \
  --efidisk0 local-zfs:1,efitype=4m,pre-enrolled-keys=1 \
  --cpu host \
  --cores 4 \
  --sockets 1 \
  --memory 8192 \
  --agent enabled=1 \
  --sata0 local-zfs:150 \
  --net0 e1000,bridge=vmbr0 \
  --ide2 local:iso/Windows_Server_2025_Evaluation_x64.iso,media=cdrom \
  --ide0 local:iso/virtio-win-0.1.302.iso,media=cdrom \
  --boot order=ide2
```

Verify the resulting configuration file on the Proxmox host:

```
# /etc/pve/qemu-server/104.conf
agent: 1
bios: ovmf
boot: order=ide2
cores: 4
cpu: host
efidisk0: local-zfs:vm-104-disk-0,efitype=4m,pre-enrolled-keys=1
ide0: local:iso/virtio-win-0.1.302.iso,media=cdrom
ide2: local:iso/Windows_Server_2025_Evaluation_x64.iso,media=cdrom
machine: q35
memory: 8192
name: WIN-CORE01
net0: e1000,bridge=vmbr0
ostype: win11
sata0: local-zfs:vm-104-disk-1,size=150G
sockets: 1
```

> **Cluster Configuration Paths:** Proxmox VE stores VM configuration files under `/etc/pve/qemu-server/<VMID>.conf`. Hardware parameters can be adjusted dynamically via `qm set` or by editing the file directly on the host shell.

Advanced Hypervisor Tuning for Production Environments

While standard settings suffice for test environments, production clusters hosting enterprise services require targeted KVM and QEMU tuning to achieve optimal I/O throughput and latency stability.

1. KVM Hyper-V Enlightenments:

By configuring `--ostype win11`, Proxmox VE exposes paravirtualized interfaces that the Windows kernel supports natively. These enlightenments present the guest with synthetic Hyper-V abstractions, drastically reducing costly VM-exits to the host:

- `hv_relaxed`: Prevents guest kernel timeouts and watchdog panics during brief host CPU scheduling spikes or overcommitment.
- `hv_spinlocks`: Optimizes guest kernel spinlock acquisition across multi-vCPU topologies.
- `hv_vapic`: Accelerates guest access to the virtualized Advanced Programmable Interrupt Controller (APIC).
- `hv_time` & `hv_synic`: Provide synthetic timer and interrupt delivery, cutting CPU overhead caused by periodic clock interrupts.

2. Storage Architecture: AIO Backends, Caching, and Dedicated IO-Threads:

When running paravirtualized `virtio-scsi-single` storage, tune latency and IOPS through three essential parameters:

```
qm set 104 \
  --scsihw virtio-scsi-single \
  --scsi0 local-zfs:64,discard=on,ssd=1,iothread=1,cache=none,aio=io_uring
```

- `iothread=1`: Allocates an independent host polling thread to the virtual SCSI controller. Disk I/O processing runs independently without blocking the primary QEMU thread responsible for CPU emulation and system timers.
- `cache=none`: Bypasses host operating system page caching via direct I/O (`O_DIRECT`). On ZFS pools, this avoids double-caching (host ZFS ARC vs. guest NTFS cache) and guarantees that writes commit transactionally to persistent storage.
- `aio=io_uring`: Utilizes the modern Linux kernel `io_uring` asynchronous I/O interface, providing noticeably lower submission latency on NVMe arrays compared to legacy POSIX `threads`.

3. vCPU Topology and NUMA Sizing:

On physical hosts equipped with multiple CPU sockets or AMD EPYC chiplet architectures, mirror the underlying NUMA (Non-Uniform Memory Access) topology into the guest:

```
qm set 104 --numa 1 --sockets 1 --cores 4
```

Without NUMA awareness, Windows treats all allocated memory as a single uniform pool. Under memory-intensive workloads, memory pages may be inadvertently routed across slower inter-socket interconnects (such as AMD Infinity Fabric or Intel UPI), degrading throughput.

4. Network Hardening and VLAN Segmentation:

In enterprise environments, management access to Server Core belongs in an isolated management VLAN. Proxmox VE allows assigning an 802.1Q VLAN tag directly to the virtual network interface without guest driver configuration:

```
qm set 104 --net0 virtio,bridge=vmbr0,tag=20,firewall=1
```

Automated VM Deployment via Shell Script

To streamline deployments, maintain an automated provisioning script on your Proxmox host. The Bash script below verifies ID availability, validates ISO paths, and creates the VM with a single invocation:

```
#!/usr/bin/env bash
# /root/deploy-wincore-vm.sh
# Automated deployment of a Windows Server 2025 Core VM on Proxmox VE 9

set -euo pipefail
```

```
VM_ID="${1:-104}"
VM_NAME="${2:-WIN-CORE01}"
STORAGE_POOL="${3:-local-zfs}"
DISK_SIZE="${4:-150G}"
RAM_MB="${5:-8192}"
CPU_CORES="${6:-4}"
BRIDGE="${7:-vmbr0}"

WIN_ISO="local:iso/Windows_Server_2025_Evaluation_x64.iso"
VIRTIO_ISO="local:iso/virtio-win-0.1.302.iso"

echo "=== Windows Server 2025 Core VM Deployment ==="
echo "VM-ID:      ${VM_ID}"
echo "Hostname:   ${VM_NAME}"
echo "Storage:    ${STORAGE_POOL} (${DISK_SIZE})"
echo "Resources:  ${CPU_CORES} Cores, ${RAM_MB} MB RAM"

if qm status "${VM_ID}" &>/dev/null; then
    echo "ERROR: VM-ID ${VM_ID} already exists on this node!" >&2
    exit 1
fi

for iso in "${WIN_ISO}" "${VIRTIO_ISO}"; do
    vol_id="${iso##*:}"
    storage_name="${iso%%:*}"
    if ! pvesm list "${storage_name}" | grep -q "${vol_id}"; then
        echo "ERROR: Image '${iso}' not found on storage '${storage_name}'!" >&2
        exit 1
    fi
done

echo "[+] ISO images verified."
echo "[+] Creating virtual machine..."
qm create "${VM_ID}" \
    --name "${VM_NAME}" \
    --ostype win11 \
    --machine q35 \
    --bios ovmf \
    --cpu host \
    --cores "${CPU_CORES}" \
    --sockets 1 \
    --memory "${RAM_MB}" \
    --numa 1 \
```

```
    --agent enabled=1

echo "[+] Configuring EFI disk and vTPM..."
qm set "${VM_ID}" --efidisk0 "${STORAGE_POOL}:1,efitype=4m,pre-enrolled-keys=1"
qm set "${VM_ID}" --tpmstate0 "${STORAGE_POOL}:1,version=v2.0"

echo "[+] Attaching virtual disks and optical media..."
qm set "${VM_ID}" \
    --sata0 "${STORAGE_POOL}:${DISK_SIZE}" \
    --net0 "e1000,bridge=${BRIDGE},firewall=1" \
    --ide2 "${WIN_ISO},media=cdrom" \
    --ide0 "${VIRTIO_ISO},media=cdrom" \
    --boot order=ide2

echo "=== Deployment completed successfully ==="
echo "Start VM using: qm start ${VM_ID}"
```

Make the script executable (`chmod +x /root/deploy-wincore-vm.sh`) and launch deployment:

```
/root/deploy-wincore-vm.sh 104 "WIN-CORE01" local-zfs 150G 8192 4 vmbr0
```

Proxmox Snapshot and Backup Strategies via CLI

Before initiating operating system installation or conducting major service updates, take advantage of Proxmox VE snapshot and backup tools:

1. ZFS-Based Snapshots before Critical Changes:

Freeze virtual machine state in seconds using `qm snapshot` :

```
# Create baseline snapshot prior to OS setup
qm snapshot 104 "clean-hw-state" --description "Clean VM hardware configuration prior to
OS installation"

# List snapshots
qm listsnapshot 104

# Roll back instantly if misconfigured
qm rollback 104 "clean-hw-state"
```

2. VSS-Integrated Backups via vzdump:

Once the QEMU Guest Agent is active inside Windows, `vzdump` generates fully application-consistent backups with VSS freeze:

```
vzdump 104 --storage local --mode snapshot --compress zstd --mailnotification failure
```

The VM is now fully provisioned on the hypervisor. Launch the instance with `qm start 104`.

Installing Windows Server 2025 Core

With the virtual machine provisioned on the Proxmox host, operating system installation begins. The initial stage is monitored via the interactive noVNC console in the Proxmox web interface because the guest has neither active networking nor remote access daemons running at this point.

The installation sequence progresses through five distinct phases:

1. **Initial Boot Phase** Power on VM, open noVNC console, and catch the CD boot key prompt
2. **Edition Selection** Choose Windows Server 2025 Standard (Server Core) as installation target
3. **Storage Controller Detection** Detect SATA immediately or load VirtIO-SCSI storage driver from attached ISO
4. **Image Extraction & First Reboot** Extract system files, install core features, and reboot into local disk
5. **Password Setting & Logon** Establish local Administrator credentials and enter SConfig

Launching the VM and Initial Boot Phase

Power on the virtual machine via the Proxmox shell or the web GUI:

```
qm start 104
```

Immediately navigate to the VM **Console** (noVNC) in the Proxmox web interface.

> **Catch the noVNC Boot Prompt Immediately:** As soon as OVMF firmware initializes, the console prompts: *„Press any key to boot from CD or DVD...“*. Click into the noVNC viewport immediately and press any key (such as the spacebar). If no input is registered within a few seconds, UEFI skips the optical drive and drops into the *UEFI Interactive Shell*. If that happens, reset the VM using the Proxmox console menu.

Once the boot environment finishes loading, the graphical Windows Setup wizard initializes. Confirm language, time and currency format, and keyboard layout, then click *Next* followed by

Install now.

Selecting the Server Core Edition

When prompted for a product key (in evaluation environments, select *„I don't have a product key“*), the setup media displays the available edition SKUs.

This is the definitive architectural fork determining system footprint:

- **Windows Server 2025 Standard** *(this represents Server Core)*
- Windows Server 2025 Standard (Desktop Experience)
- Windows Server 2025 Datacenter
- Windows Server 2025 Datacenter (Desktop Experience)

> ⚠ **Edition Naming in Setup Media:** Microsoft designates the headless Server Core variant simply as *Windows Server 2025 Standard*—without appending „Server Core“. The full graphical environment is explicitly labeled *(Desktop Experience)*. Select the first option to deploy Server Core.

Accept the license terms, then choose the custom installation path: **Custom: Install Microsoft Windows Server only (advanced)**.

Disk Detection: SATA vs. VirtIO-SCSI

In the *„Where do you want to install Windows?“* storage dialog, the workflow depends directly on the storage controller selected during VM creation:

- **SATA Storage (Our Documented Lab Configuration):**

Windows Server 2025 includes inbox AHCI controllers. The 150 GiB virtual disk appears immediately as *„Drive 0 Unallocated Space“*. No driver loading is required. Click *Next* to proceed directly.

- **VirtIO-SCSI Storage (Recommended Production Alternative):**

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

The disk list is initially completely empty because Windows does not package paravirtualized KVM storage drivers out of the box. Execute the following steps:

1. Click **Load driver**, then click **Browse**.
2. Navigate to the secondary optical drive housing the attached `virtio-win` ISO.
3. Expand the directory `vioscsi` → `2k25` → `amd64`.
4. Select the driver: *Red Hat VirtIO SCSI pass-through controller*.
5. Upon confirmation, the paravirtualized drive appears in the partition list.

> **Automatic UEFI Partition Allocation:** Clicking *Next* on unallocated disk space prompts Windows Setup to automatically partition the drive according to UEFI standards: allocating an EFI System Partition (FAT32, ~100 MiB), a Microsoft Reserved Partition (MSR, 16 MiB), and the primary operating system partition (NTFS).

The installer extracts the WIM image, configures boot loaders, and initiates an automated system restart within minutes.

Setting Administrator Password and Initial Logon

Following the reboot, the virtual machine boots directly from its virtual hard drive. In place of a graphical welcome wizard, Server Core initializes a text-based console prompt displaying a blue dialog overlay:

```
The user's password must be changed before signing in.
```

1. Press `Enter` on the preselected `<OK>` field.
2. Enter the new password for the local `Administrator` account, then repeat it on the confirmation line.
3. Confirm by pressing `Enter`.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

> ⚠ **Mind Keyboard Layout in noVNC:** Depending on browser locale settings and hypervisor mappings, the noVNC console may default to a standard US layout (QWERTY) prior to operating system configuration. Take special care when typing passwords containing transposed characters (`Z` and `Y`) or punctuation to avoid authentication lockouts during subsequent SSH sessions.

Windows confirms with the dialog *„Your password has been changed“*. After pressing `Enter`, the system logs in the Administrator.

A black command prompt window (`cmd.exe`) appears, followed seconds later by the automatic launch of the blue **SConfig** console menu in the foreground. The system has reached its operational baseline.

Fully Automated Unattended Deployment with autounattend.xml

In enterprise datacenters or automated lab pipelines, manual clicking through noVNC is inefficient. Windows Server 2025 supports fully automated zero-touch provisioning using an answer file named `autounattend.xml`.

When placed in the root directory of an attached virtual volume (such as a secondary virtual CD-ROM or virtual floppy disk), Windows Setup bypasses interactive prompts, automatically partitions the drive, selects the Core SKU, provisions the Administrator account, and executes initial setup commands.

1. Structure of the autounattend.xml Answer File:

The following XML configuration is optimized for automated UEFI deployments of Windows Server 2025 Standard (Core):

```
<?xml version="1.0" encoding="utf-8"?>
<unattend xmlns="urn:schemas-microsoft-com:unattend">
```

```
  <!-- Phase 1: Windows Preinstallation Environment (WinPE) -->
  <settings pass="windowsPE">
    <component name="Microsoft-Windows-International-Core-WinPE"
processorArchitecture="amd64" publicKeyToken="31bf3856ad364e35" language="neutral"
versionScope="nonSxS">
      <SetupUILanguage>
        <UILanguage>en-US</UILanguage>
      </SetupUILanguage>
      <InputLocale>en-US</InputLocale>
      <SystemLocale>en-US</SystemLocale>
      <UILanguage>en-US</UILanguage>
      <UserLocale>en-US</UserLocale>
    </component>
    <component name="Microsoft-Windows-Setup" processorArchitecture="amd64"
publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS">
      <DiskConfiguration>
        <Disk wcm:action="add"
xmlns:wcm="http://schemas.microsoft.com/WMIConfig/2002/State">
          <DiskID>0</DiskID>
          <WillWipeDisk>true</WillWipeDisk>
          <CreatePartitions>
            <!-- 1. EFI System Partition (FAT32, 100 MB) -->
            <CreatePartition wcm:action="add">
              <Order>1</Order>
              <Type>EFI</Type>
              <Size>100</Size>
            </CreatePartition>
            <!-- 2. Microsoft Reserved Partition (MSR, 16 MB) -->
            <CreatePartition wcm:action="add">
              <Order>2</Order>
              <Type>MSR</Type>
              <Size>16</Size>
            </CreatePartition>
            <!-- 3. Primary OS Partition (Rest of disk) -->
            <CreatePartition wcm:action="add">
              <Order>3</Order>
              <Type>Primary</Type>
              <Extend>true</Extend>
            </CreatePartition>
          </CreatePartitions>
          <ModifyPartitions>
            <ModifyPartition wcm:action="add">
              <Order>1</Order>
              <PartitionID>1</PartitionID>
```

```
            <Format>FAT32</Format>
            <Label>System</Label>
          </ModifyPartition>
          <ModifyPartition wcm:action="add">
            <Order>2</Order>
            <PartitionID>2</PartitionID>
          </ModifyPartition>
          <ModifyPartition wcm:action="add">
            <Order>3</Order>
            <PartitionID>3</PartitionID>
            <Format>NTFS</Format>
            <Label>Windows</Label>
            <Letter>C</Letter>
          </ModifyPartition>
        </ModifyPartitions>
      </Disk>
    </DiskConfiguration>
    <ImageInstall>
      <OSImage>
        <InstallFrom>
          <MetaData wcm:action="add">
            <Key>/IMAGE/NAME</Key>
            <!-- Exact SKU string for Server Core in Windows Server 2025 -->
            <Value>Windows Server 2025 Standard</Value>
          </MetaData>
        </InstallFrom>
        <InstallTo>
          <DiskID>0</DiskID>
          <PartitionID>3</PartitionID>
        </InstallTo>
      </OSImage>
    </ImageInstall>
    <UserData>
      <AcceptEula>true</AcceptEula>
    </UserData>
  </component>
</settings>

<!-- Phase 2: System Specialization -->
<settings pass="specialize">
  <component name="Microsoft-Windows-Shell-Setup" processorArchitecture="amd64"
publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS">
    <ComputerName>WIN-CORE01</ComputerName>
    <TimeZone>W. Europe Standard Time</TimeZone>
```

```
    </component>
  </settings>

  <!-- Phase 3: Out-of-Box Experience (OOBE) & First Logon Execution -->
  <settings pass="oobeSystem">
    <component name="Microsoft-Windows-Shell-Setup" processorArchitecture="amd64"
publicKeyToken="31bf3856ad364e35" language="neutral" versionScope="nonSxS">
      <UserAccounts>
        <AdministratorPassword>
          <Value>Start12345!</Value>
          <PlainText>true</PlainText>
        </AdministratorPassword>
      </UserAccounts>
      <!-- Synchronous commands executed at initial automated logon -->
      <FirstLogonCommands>
        <SynchronousCommand wcm:action="add">
          <Order>1</Order>
          <Description>Install OpenSSH Server Capability</Description>
          <CommandLine>powershell.exe -Command "Add-WindowsCapability -Online -Name
OpenSSH.Server~~~~0.0.1.0"</CommandLine>
        </SynchronousCommand>
        <SynchronousCommand wcm:action="add">
          <Order>2</Order>
          <Description>Enable and Start OpenSSH Service</Description>
          <CommandLine>powershell.exe -Command "Set-Service -Name sshd -StartupType
Automatic; Start-Service -Name sshd"</CommandLine>
        </SynchronousCommand>
        <SynchronousCommand wcm:action="add">
          <Order>3</Order>
          <Description>Disable SConfig Autolaunch</Description>
          <CommandLine>powershell.exe -Command "Set-SConfig -AutoLaunch
$false"</CommandLine>
        </SynchronousCommand>
      </FirstLogonCommands>
    </component>
  </settings>
</unattend>
```

> **Security Risk of Plaintext Credentials in Answer Files:** Setting `<PlainText>true</PlainText>` stores the administrator password in unencrypted

> plaintext within the XML structure. Any user or process with read permissions to the ISO media, virtual storage volume, or `autounattend.xml` file can extract the credentials directly. A subsequent password change after initial login does not retroactively remediate this exposure, as the answer file remains on the installation medium. The string `Start12345!` is strictly a non-production example placeholder. For real deployments, provision individual temporary credentials via controlled deployment workflows and rotate them immediately. The Microsoft-documented setting `<PlainText>false</PlainText>` merely obfuscates the value within the file and does not constitute cryptographically secure encryption.

2. Packaging the Answer File into an ISO on Proxmox:

To present `autounattend.xml` to WinPE during boot, package it into a minimal optical image on the Proxmox host:

```
# Create working directory and generate XML file
mkdir -p /tmp/unattend
cat <<'EOF' > /tmp/unattend/autounattend.xml
<!-- Insert XML content from above -->
EOF

# Build ISO in local template storage
genisoimage -o /var/lib/vz/template/iso/unattend-wincore.iso -J -r /tmp/unattend/
```

3. Attaching the Answer ISO for Zero-Touch Deployment:

Attach the generated ISO to an available optical drive interface on the VM:

```
qm set 104 --ide1 local:iso/unattend-wincore.iso,media=cdrom
```

Starting the VM with `qm start 104` triggers unattended installation. Without a single console interaction, Windows partitions storage, deploys Server Core, configures credentials, and activates OpenSSH. Within minutes, the system becomes accessible directly via SSH from your Linux workstation.

Device Inspection and Driver Installation without Device Manager (pnputil)

Server Core does not provide the traditional Device Manager graphical snap-in (`devmgmt.msc`). To audit driver health and confirm virtual KVM hardware (such as storage adapters, ballooning drivers, and network cards) initialized cleanly, rely on PowerShell and Microsoft's native `pnputil.exe` utility.

1. Inspecting Devices with Errors or Missing Drivers:

Use `Get-PnpDevice` to query components whose status diverges from `OK`:

```
Get-PnpDevice | Where-Object Status -ne 'OK' | Format-Table -Property InstanceId, Class,
FriendlyName, Status
```

Components lacking paravirtualized drivers will be flagged with a status of `Error` or `Unknown`.

2. Installing All Drivers from the VirtIO ISO in a Single Command:

With the `virtio-win` ISO mounted to drive `D:`, install all paravirtualized drivers recursively across subdirectories:

```
pnputil.exe /add-driver D:\*.inf /subdirs /install
```

`pnputil.exe` parses all driver packages, imports signatures into the Windows Driver Store, and binds drivers directly to detected hardware.

3. Triggering Hardware Rescan and Verifying Active Drivers:

Force the PCI bus to re-enumerate attached hardware, then audit active VirtIO components:

```
# Force device bus scan
pnputil.exe /scan-devices

# Verify VirtIO and QEMU devices
Get-PnpDevice -Class Net, SCSIAdapter, System |
    Where-Object { $_.FriendlyName -match 'Red Hat|VirtIO|QEMU' } |
    Format-Table -Property Class, FriendlyName, Status
```

Initial Configuration with SConfig

Immediately following first logon, Server Core presents the menu-driven **SConfig** utility (`sconfig.cmd`). Rewritten entirely on top of PowerShell in modern Windows Server versions, SConfig aggregates essential post-install administrative tasks required before connecting the machine to production networks.

Navigation is numeric: enter the number of the desired task, press `Enter` , and answer the prompt.

Overview of the SConfig Main Menu

The Windows Server 2025 SConfig menu organizes baseline setup into 15 structured options:

Option	Menu Item	Default Status / Operational Function
1	Domain/Workgroup	`Workgroup: WORKGROUP` (or initiate domain join)
2	Computer Name	`WIN-CORE01` (NetBIOS and DNS hostname)
3	Add Local Administrator	Create secondary emergency local accounts
4	Configure Remote Management	`Enabled` (WinRM, Server Manager, and WAC endpoints)
5	Windows Update Settings	`DownloadOnly` (or Automatic / Manual)
6	Download and Install Updates	Immediately query and apply security patches
7	Remote Desktop	`Disabled` (or Enabled with Network Level Authentication)
8	Network Settings	Configure IP addressing, subnet masks, gateways, and DNS
9	Date and Time	Set timezone and NTP synchronization
10	Telemetry Settings	`Required` (diagnostic telemetry level)
11	Windows Activation	Audit activation status or input license keys
12	Log Off User	End active console session
13	Restart Server	Perform controlled operating system reboot
14	Shut Down Server	Perform clean guest powerdown
15	Exit to Command Line (PowerShell)	Exit SConfig and return to native PowerShell prompt

> **Controlling SConfig Autolaunch:** By default, SConfig launches automatically upon every interactive logon. To drop directly into PowerShell instead, disable this behavior permanently using `Set-SConfig -AutoLaunch $false` (re-enable with `$true`). SConfig remains accessible on demand by entering `sconfig`.

Setting Computer Name and Workgroup

Windows Setup defaults to generating random hostnames such as `WIN-7K2M9LPQ3R1`. Standardize this according to your datacenter naming conventions:

1. Select Option `2` (*Computer Name*) and press `Enter`.
2. Input the new hostname (in our documented test VM: `WIN-CORE01`).
3. SConfig asks whether to reboot immediately. Choose `No (N)` to finalize network addressing within the same session.

Option `1` (*Domain/Workgroup*) remains configured to workgroup `WORKGROUP` for this guide.

Network Configuration: Static IPv4 and DNS Resolvers

Windows Server Core defaults to DHCP addressing. Production server operation requires static assignment:

1. Select Option `8` (*Network Settings*).
2. SConfig displays all detected network interfaces along with their interface index. The active network card typically maps to index `1`.
3. Input the **interface index** (e.g., `1`) and confirm with `Enter`.

The interface configuration submenu opens:

- **Assign Static IP Address:**

1. Select Option `1` (*Set Network Adapter Address*).
2. In response to the addressing type prompt, enter `S` (*Static*).
3. Input your static IP (for our test lab: `192.0.2.10`).
4. Input the subnet mask (`255.255.255.0` for a `/24` prefix).
5. Input default gateway address: `192.0.2.1`.

- **Configure DNS Resolvers:**

1. Select Option `2` (*Set DNS Servers*).
2. Input the primary IPv4 address of your local DNS resolver (e.g., `192.0.2.1`).
3. Specify a secondary resolver or press `Enter` to skip.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

- **Return to Main Menu:** Select Option `4` to navigate back to the main menu.

Remote Management, Remote Desktop, and Windows Updates

Configure remaining baseline subsystems before proceeding:

- **Option 4 (Configure Remote Management):** Confirm Remote Management is set to *Enabled*, provisioning local WinRM listeners for Windows Admin Center and RSAT tools.
- **Option 7 (Remote Desktop):** Optional text console RDP access can be enabled if required (enter `E`, followed by Option `1` for *Network Level Authentication / NLA*).

> **RDP on Server Core Lacks Desktop Shell:** Connecting via an RDP client (such as FreeRDP, Remmina, or mstsc) to Server Core opens an empty full-screen window displaying a single command prompt (`cmd.exe`). For Linux administrators, OpenSSH over standard terminals provides vastly superior terminal emulation and automation capabilities.

- **Options 5 & 6 (Windows Updates):** Set the update behavior under Option 5 to *Manual* (`M`) or *DownloadOnly* (`D`) to prevent unexpected background downloads during setup. If outbound internet connectivity is established, Option 6 allows downloading current security patches immediately.
- **Option 11 (Windows Activation):** Evaluation editions provide 180 days of fully functional operation without key entry.

Orderly Reboot and Transition to PowerShell

Reboot the guest to ensure all network stacks, NetBIOS bindings, and security identifiers (SIDs) initialize under the new hostname:

1. In the SConfig menu, select Option `13` (*Restart Server*).
2. Confirm the reboot prompt with `Yes` (`Y`).

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

> ⚠ **Reboot Prior to OpenSSH Deployment:** Do not skip this reboot. If OpenSSH is initialized prior to updating the computer name, the service generates host certificates and firewall bindings tied to the temporary random hostname. The reboot guarantees the system publishes its proper `WIN-CORE01` identity across the subnet.

Following the reboot, log on via noVNC as `Administrator`. When SConfig loads, exit to the command line:

Select `15` (*Exit to Command Line (PowerShell)*).

SConfig terminates, leaving you at a native PowerShell prompt.

SConfig vs. Native PowerShell: Complete Reference Matrix

For automated administration and infrastructure-as-code workflows (such as Ansible playbooks), SConfig serves as an interactive wrapper around native PowerShell cmdlets and WMI APIs:

No.	SConfig Menu Item	Native PowerShell Cmdlet / API Equivalent
1	Domain/Workgroup	`Add-Computer -WorkgroupName "WORKGROUP"` or `-DomainName "corp.example.com"`
2	Computer Name	`Rename-Computer -NewName "WIN-CORE01" -Force -Restart`
3	Add Local Administrator	`New-LocalUser -Name "secadmin" -Password $Password; Add-LocalGroupMember -Group "Administrators" -Member "secadmin"`
4	Configure Remote Management	`Configure-SMRemoting.exe -enable` or `Enable-PSRemoting -Force`
5	Windows Update Settings	`(New-Object -ComObject "Microsoft.Update.AutoUpdate").Settings.NotificationLevel = 2`
6	Download and Install Updates	`Install-WindowsUpdate -MicrosoftUpdate -AcceptAll -AutoReboot`

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

No.	SConfig Menu Item	Native PowerShell Cmdlet / API Equivalent
7	Remote Desktop (RDP)	`Set-ItemProperty -Path 'HKLM:\System\CurrentControlSet\Control\Terminal Server' -Name 'fDenyTSConnections' -Value 0`
8	Network Settings	`New-NetIPAddress -InterfaceAlias 'Ethernet0' -IPAddress 192.0.2.10 -PrefixLength 24 -DefaultGateway 192.0.2.1`
9	Date and Time	`Set-TimeZone -Id "W. Europe Standard Time"; w32tm /resync`
10	Telemetry Settings	`Set-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\DataCollection' -Name 'AllowTelemetry' -Value 1`
11	Windows Activation	`slmgr.vbs /ipk <ProductKey>; slmgr.vbs /ato`
12	Log Off User	`logoff.exe`
13	Restart Server	`Restart-Computer -Force`
14	Shut Down Server	`Stop-Computer -Force`
15	Exit to Command Line	`exit`

Automated Post-Deployment via PowerShell Script

The manual SConfig sequence can be encapsulated into an automated post-deployment script:

□□ Praktisches Beispiel: Initialization Script Initialize-ServerCore.ps1

Execute this script at the PowerShell console immediately upon initial logon:

```
# Initialize-ServerCore.ps1
# Automated headless baseline configuration for Windows Server 2025 Core
```

```
[CmdletBinding()]
param (
    [string]$NewComputerName = "WIN-CORE01",
    [string]$InterfaceAlias = "Ethernet0",
    [string]$StaticIP = "192.0.2.10",
    [int]$PrefixLength = 24,
    [string]$Gateway = "192.0.2.1",
    [string[]]$DnsServers = @("192.0.2.1", "1.1.1.1"),
    [string]$TimeZoneId = "W. Europe Standard Time"
)

Write-Host "=== Starting Automated Baseline Configuration ===" -ForegroundColor Cyan

# 1. Configure Timezone
Write-Host "[1/6] Setting timezone to $TimeZoneId..." -ForegroundColor Yellow
Set-TimeZone -Id $TimeZoneId

# 2. Disable SConfig Autostart
Write-Host "[2/6] Disabling automated SConfig startup..." -ForegroundColor Yellow
Set-SConfig -AutoLaunch $false

# 3. Configure Static IP and Default Gateway
Write-Host "[3/6] Configuring network adapter $InterfaceAlias
($StaticIP/$PrefixLength)..." -ForegroundColor Yellow
Get-NetIPAddress -InterfaceAlias $InterfaceAlias -AddressFamily IPv4 -ErrorAction
SilentlyContinue |
    Remove-NetIPAddress -Confirm:$false

New-NetIPAddress -InterfaceAlias $InterfaceAlias -IPAddress $StaticIP -PrefixLength
$PrefixLength -DefaultGateway $Gateway

# 4. Assign DNS Resolvers
Write-Host "[4/6] Registering DNS resolvers..." -ForegroundColor Yellow
Set-DnsClientServerAddress -InterfaceAlias $InterfaceAlias -ServerAddresses $DnsServers

# 5. Set Telemetry to Minimum (Required / Value 1)
Write-Host "[5/6] Restricting diagnostic data telemetry..." -ForegroundColor Yellow
Set-ItemProperty -Path
'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\DataCollection' -Name
'AllowTelemetry' -Value 1 -Type DWord

# 6. Apply Computer Name and Initiate Reboot
$CurrentName = $env:COMPUTERNAME
```

```
if ($CurrentName -ne $NewComputerName) {
    Write-Host "[6/6] Renaming system from '$CurrentName' to '$NewComputerName'..." -
ForegroundColor Yellow
    Rename-Computer -NewName $NewComputerName -Force
    Write-Host "=== Configuration complete. Rebooting in 5 seconds... ===" -
ForegroundColor Green
    Start-Sleep -Seconds 5
    Restart-Computer -Force
} else {
    Write-Host "=== Configuration complete. No reboot required. ===" -ForegroundColor
Green
}
```

Configuring Locale, Region, and Keyboard Layout via PowerShell

Server Core lacks the graphical Control Panel applet for regional settings. If keyboard mappings or system locales diverge from your requirements, adjust them using the `International` module:

1. Inspecting Current Locale Settings:

```
Get-WinUserLanguageList | Format-Table -Property LanguageTag, Autonym, EnglishName
Get-Culture | Format-Table -Property Name, DisplayName
Get-WinSystemLocale | Format-Table -Property Name, DisplayName
```

2. Standardizing Regional Defaults via PowerShell:

```
# Set English (US) or German layout as needed
$LangList = New-WinUserLanguageList -Language "en-US"
Set-WinUserLanguageList -LanguageList $LangList -Force

# System locale for non-Unicode programs
```

```
Set-WinSystemLocale -SystemLocale "en-US"

# Regional formatting culture
Set-Culture -CultureInfo "en-US"

# Set GeoID (e.g., United States = 244, Germany = 94)
Set-WinHomeLocation -GeoId 244
```

Following logon, language configurations take effect. The server is primed for OpenSSH deployment.

Installing and Configuring OpenSSH Server

With SConfig dismissed, we transition management entirely to the network. To manage Windows Server Core permanently from Linux terminals without opening hypervisor consoles, we now configure the integrated OpenSSH daemon.

Unlike WinRM, which relies on HTTP/SOAP and demands complex HTTPS certificates for mutual server authentication or TrustedHosts exceptions without identity validation outside Active Directory domains, OpenSSH adheres to industry-standard transport mechanics: encrypted tunnels over TCP port 22, host-key verification, and robust public-key cryptography.

The deployment sequence proceeds through four phases:

1. **Audit Status** Verify OpenSSH server capability and runtime status via PowerShell
2. **Enable & Start Daemon** Set startup type to Automatic and launch sshd service
3. **Verify Firewall & Profile** Audit port 22 firewall bindings and check network profile
4. **Establish Remote Access** Connect via ssh from Linux and configure PowerShell as default shell

Verifying Capability and Installation Status

Under Windows Server 2025, OpenSSH Server is integrated directly into the operating system image. Verify package presence using `Get-WindowsCapability`:

```
Get-WindowsCapability -Online -Name OpenSSH.Server*
```

If the state reports `State : Installed`, the package is fully available.

If deploying on a custom image where the component is `NotPresent`, install it using PowerShell:

```
Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
```

Service Startup, Automated Launch, and Status Auditing

For security, the `sshd` service is set to `Stopped` with a `Manual` startup type after installation.

1. Configure Automatic Startup:

```
Set-Service -Name sshd -StartupType Automatic
```

2. Launch OpenSSH Service:

```
Start-Service -Name sshd
```

3. Audit Service Status and Listening Socket:

Confirm the service is running and listening locally on port 22:

```
Get-Service -Name sshd | Format-Table -Property Name, DisplayName, Status, StartType
```

Expected output:

```
Name Status  StartType DisplayName
---- ------  --------- -----------
sshd Running Automatic OpenSSH SSH Server
```

Verify port 22 socket binding:

```
Get-NetTCPConnection -LocalPort 22 -State Listen | Format-Table -Property LocalAddress,
LocalPort, State
```

> **Host Key Generation on Initial Launch:** When `sshd` starts for the first time, Windows OpenSSH automatically generates host key pairs (RSA, ECDSA, and Ed25519) and stores them in the protected path `C:\ProgramData\ssh\`.

Firewall Rules and Windows Network Profiles

To ensure incoming SSH connections are not dropped by Windows Defender Firewall, firewall rules and network connection profiles must align.

Windows segments interfaces into three security profiles:

- **`Public`:** Default profile for newly detected adapters. Blocks incoming unsolicited traffic and disables network discovery.
- **`Private`:** Designed for trusted internal segments behind physical firewalls. Permits administrative shares and ICMP echo requests.
- **`DomainAuthenticated`:** Activates automatically once the server joins an Active Directory domain and verifies domain controller connectivity.

Audit your current network profile:

```
Get-NetConnectionProfile
```

> ⚠ **Security Implications of Profile Transitions:** Switching an adapter from `Public` to `Private` opens other Windows services (such as SMB shares and NetBIOS) across the subnet. Only switch to `Private` if the server is isolated within a dedicated management VLAN. In shared environments, leave the profile set to `Public` and bind the SSH firewall rule specifically.

For isolated lab environments, set the profile to Private via PowerShell:

```
Set-NetConnectionProfile -InterfaceAlias "Ethernet" -NetworkCategory Private
```

Audit the inbound firewall rule for OpenSSH:

```
Get-NetFirewallRule -Name *OpenSSH-Server* | Format-Table -Property Name, DisplayName,
Enabled, Profile, Action
```

To ensure connectivity regardless of active profile, bind the rule across all profiles:

```
Set-NetFirewallRule -Name "OpenSSH-Server-In-TCP" -Profile Any -Enabled True
```

Establishing Remote SSH Connectivity from Linux

Initiate terminal access from your Linux workstation using standard OpenSSH, targeting the configured static IP (`192.0.2.10`):

```
ssh Administrator@192.0.2.10
```

1. **Verify Host Key Fingerprint:**

During initial connection, the client displays the host key fingerprint. Confirm by entering `yes` .

1. **Authenticate:**

Input the password established during Windows setup.

Upon authentication, Windows spawns a command shell. You are now connected to Windows Server 2025 Core via SSH—noVNC is no longer needed.

Setting PowerShell as Default Shell in Registry

By default, Windows OpenSSH spawns a legacy command prompt (`cmd.exe`), displaying:

```
C:\Users\Administrator>
```

While entering `powershell` transitions to the modern shell, configuring OpenSSH to launch PowerShell natively provides a vastly superior terminal workflow.

Set the `DefaultShell` value in the Windows Registry:

```
New-ItemProperty -Path "HKLM:\SOFTWARE\OpenSSH" `
  -Name DefaultShell `
  -Value "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" `
  -PropertyType String -Force
```

Disconnect via `exit` and reconnect from Linux. The session lands immediately in native PowerShell:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator>
```

Key-Based Authentication for Administrators

Password authentication is fragile and unsuitable for automated Ansible playbooks or CI/CD pipelines.

Under Windows OpenSSH, local administrators adhere to a specific security architecture that diverges from Linux conventions:

- **Standard Windows Users:** Utilize user profile files at `C:\Users\<Username>\.ssh\authorized_keys`.
- **Local Administrators:** Windows strictly ignores profile-based keys and requires placing public keys into a centralized administrative file:

`C:\ProgramData\ssh\administrators_authorized_keys`

> **Strict ACL Requirements for administrators_authorized_keys:** The OpenSSH daemon rejects public-key authentication if `administrators_authorized_keys` contains permissive ACLs. Only `SYSTEM` and `BUILTIN\Administrators` may possess read or write rights; all inherited permissions must be stripped.

Praktisches Beispiel: Deploying SSH Public Keys from Linux

Append your Linux client's public key (e.g., `~/.ssh/id_ed25519.pub`) and configure NTFS Access Control Lists (ACLs) using `icacls.exe`:

```
$PublicKey = "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIG... user@linux-workstation"
$AuthKeysPath = "$env:ProgramData\ssh\administrators_authorized_keys"

# Write public key to file
Set-Content -Path $AuthKeysPath -Value $PublicKey -Encoding Ascii

# Strip inheritance and grant exclusive Full Control to SYSTEM and Administrators
icacls.exe $AuthKeysPath /inheritance:r /grant "NT AUTHORITY\SYSTEM:(F)" /grant
"BUILTIN\Administrators:(F)"
```

Test passwordless authentication from your Linux workstation:

```
ssh -i ~/.ssh/id_ed25519 Administrator@192.0.2.10
```

Authentication completes instantly without credential prompts.

Hardening the SSH Daemon (sshd_config)

The central configuration file resides at `C:\ProgramData\ssh\sshd_config`. Once key-based authentication is verified, harden the daemon against brute-force attacks and legacy ciphers.

1. Disable Password Authentication:

Update `sshd_config` to enforce public-key authentication:

```
$ConfigPath = "$env:ProgramData\ssh\sshd_config"
$Config = Get-Content $ConfigPath
```

```
$Config = $Config -replace '^#?PasswordAuthentication\s+.*', 'PasswordAuthentication no'
$Config = $Config -replace '^#?PubkeyAuthentication\s+.*', 'PubkeyAuthentication yes'
$Config = $Config -replace '^#?LogLevel\s+.*', 'LogLevel VERBOSE'

Set-Content -Path $ConfigPath -Value $Config -Encoding Ascii
```

2. Restrict Access via AllowUsers:

Prevent unauthorized accounts from negotiating SSH sessions:

```
Add-Content -Path $ConfigPath -Value "`nAllowUsers Administrator secadmin"
```

3. Enforce Modern Cryptography and Post-Quantum Hybrid KEX:

Windows Server 2025 supports post-quantum hybrid key exchange algorithms (`sntrup761`). Enforce robust ciphers and disable weak MACs:

```
# Hardened cryptographic parameters for Windows Server 2025 OpenSSH
KexAlgorithms sntrup761x25519-sha512@openssh.com,curve25519-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com
```

4. Validate Syntax and Restart Service:

Always test configuration syntax before restarting the daemon:

```
# Validate syntax
& "C:\Windows\System32\OpenSSH\sshd.exe" -t

# Restart daemon upon success
Restart-Service sshd
```

> **Maintain an Active Terminal Session:** Never terminate your active SSH session before verifying daemon configuration. Open a secondary terminal on your Linux client and confirm access: `ssh -i ~/.ssh/id_ed25519 Administrator@192.0.2.10`. Only terminate the primary session after the test login succeeds.

Diagnosing SSH Connection Failures via Windows Event Logs

OpenSSH records runtime telemetry and failed authentications directly into a dedicated Windows Event Log: `OpenSSH/Operational`.

If connections fail (such as invalid permissions on `administrators_authorized_keys`), inspect the log via PowerShell:

Praktisches Beispiel: Real-Time OpenSSH Event Log Auditing

```
# Retrieve recent OpenSSH events
Get-WinEvent -LogName "OpenSSH/Operational" -MaxEvents 20 |
    Select-Object TimeCreated, Id, LevelDisplayName, Message |
    Format-Table -Wrap
```

Key Event IDs:

- **Event ID 4 (Error):** *„Authentication refused: bad ownership or modes for file...“*

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

$\rightarrow$ File permissions on `administrators_authorized_keys` are too open. Re-apply the `icacls` command above.

- **Event ID 130 (Information):** *„Accepted key RSA/ED25519...“*

$\rightarrow$ Successful public-key authentication.

- **Event ID 131 (Information):** *„Failed publickey for Administrator...“*

$\rightarrow$ Presented client public key is not registered in `administrators_authorized_keys` or the key type was rejected.

Secure File Transfers via SCP and SFTP

WinRM struggles with transferring large binary payloads (such as ISOs, patches, or setup packages). OpenSSH includes native SFTP and SCP subsystems powered by `sftp-server.exe`:

```
Subsystem sftp sftp-server.exe
```

Copying Files from Linux to Windows via SCP:

Standard SCP syntax applies. Specify Windows destinations using forward slashes (`/`):

```
# Transfer single PowerShell script to C:\Scripts
scp -i ~/.ssh/id_ed25519 setup-app.ps1 Administrator@192.0.2.10:C:/Scripts/

# Recursively transfer entire installation directory
scp -i ~/.ssh/id_ed25519 -r ./install-packages/ Administrator@192.0.2.10:C:/Setup/
```

Interactive SFTP Sessions:

```
sftp -i ~/.ssh/id_ed25519 Administrator@192.0.2.10
```

Inside the SFTP session:

```
Connected to 192.0.2.10.
sftp> pwd
Remote path: /C:/Users/Administrator
sftp> cd C:/Shares/Deploy
sftp> put package.zip
Uploading package.zip to /C:/Shares/Deploy/package.zip
package.zip                                   100%  142MB  48.2MB/s   00:02
sftp> bye
```

SSH Local Port Forwarding for Internal Services

Local Port Forwarding securely tunnels internal administrative services or web interfaces bound to `127.0.0.1` back to your local workstation through the encrypted SSH tunnel:

□□ Praktisches Beispiel: Secure Port Tunneling over SSH

```
# Forward local port 8080 to WIN-CORE01 port 80
ssh -i ~/.ssh/id_ed25519 -L 8080:localhost:80 Administrator@192.0.2.10
```

Navigating to `http://localhost:8080` in your local browser accesses the server's internal HTTP service without exposing port 80 to the network.

Administering Windows Server Core with PowerShell

Once authenticated via SSH or logging into the hypervisor console, you interact directly with the PowerShell prompt. On Windows Server Core, the shell is not an auxiliary troubleshooting utility—it is the primary interface for diagnostics, system configuration, storage layout, and service orchestration.

Windows PowerShell 5.1 vs. PowerShell 7 (Core)

Windows Server 2025 includes inbox **Windows PowerShell 5.1**. This version runs on top of the classic .NET Framework 4.8 and is deeply coupled with the underlying operating system and native WMI/CIM infrastructure modules.

In parallel, Microsoft maintains **PowerShell 7** (formerly PowerShell Core), which builds on modern .NET (Core) runtimes, offers cross-platform parity, and ships via out-of-band release cadences.

Windows PowerShell 5.1

Runtime Engine
: .NET Framework 4.8

Installation Model
: Built directly into OS media

Module Compatibility
: Comprehensive native role coverage

Typical Use Case
: Headless OS baseline operations

PowerShell 7 (Core)

Runtime Engine

.NET 8 / 9

Installation Model

Optional package (MSI or winget)

Module Compatibility

High (partially via WinPS compatibility layer)

Typical Use Case

Modern DevOps, cross-platform CI/CD, cloud

Common: Identical cmdlet naming paradigms, pipeline mechanics, and object structures

For foundational Server Core administration, pre-installed Windows PowerShell 5.1 is completely sufficient. Every cmdlet, module, and automation script demonstrated throughout this guide operates natively out of the box without requiring external framework installations.

Understanding the Pipeline Object Model

The defining difference between Unix shells (such as Bash or Zsh) and PowerShell lies in pipeline data transport: while Unix pipelines pass raw, unformatted byte streams (text) requiring string parsing via `grep`, `awk`, or `cut`, the PowerShell pipeline carries **strongly typed .NET objects**.

Every object exposes properties and methods. Filtering, sorting, and projection operations query these typed data fields directly without string manipulation or regex parsing.

Essential pipeline control cmdlets:

- `Where-Object` (alias `?`): Filters streaming objects against conditional expressions.
- `Select-Object` (alias `select`): Projects specific properties or limits result counts (`-First`, `-Last`).
- `Sort-Object` (alias `sort`): Sorts objects ascending or descending across one or more properties.

- `Measure-Object` (alias `measure`): Computes counts, sums, averages, and statistical bounds on numeric properties.
- `Format-Table` (alias `ft`) and `Format-List` (alias `fl`): Format objects into visual representations for human terminal consumption.

> ⚠ **Pipeline Formatting Boundary Rule:** Cmdlets like `Format-Table` and `Format-List` dismantle underlying .NET objects, replacing them with visual layout control tokens. Always place `Format-*` cmdlets at the absolute end of a pipeline chain, and never pipe formatted output into subsequent filtering or transformation cmdlets.

□□ Praktisches Beispiel: Object-Based Filtering vs. String Parsing

Querying service status illustrates this distinction clearly. Rather than parsing raw terminal strings, filter directly against the typed `Status` property:

```
Get-Service | Where-Object Status -eq 'Running' | Select-Object -Property Name,
DisplayName, StartType | Sort-Object Name
```

Expected output:

```
Name     DisplayName                     StartType
----     -----------                     ---------
BFE      Base Filtering Engine           Automatic
Dhcp     DHCP Client                     Automatic
EventLog Windows Event Log               Automatic
LanmanServer Server                      Automatic
sshd     OpenSSH SSH Server              Automatic
```

```
w32time  Windows Time                         Automatic
WinRM    Windows Remote Management (WS-Man)  Automatic
```

System and Network Inspection

Operating headless requires querying baseline host metrics quickly and reliably via the CLI.

System Status, Platform Version, and Uptime:

Use `Get-ComputerInfo` to query hardware and operating system parameters:

```
Get-ComputerInfo | Select-Object -Property CsName, WindowsProductName, WindowsVersion,
OsUptime, OsLastBootUpTime, CsTotalPhysicalMemory
```

For rapid auditing of boot timestamps and available physical RAM, query WMI/CIM directly via `Get-CimInstance`:

```
Get-CimInstance -ClassName Win32_OperatingSystem | Select-Object -Property Caption,
LastBootUpTime, @{Name='FreeRAM_MB'; Expression={[math]::Round($_.FreePhysicalMemory /
1KB, 0)}}
```

Auditing Network Configuration:

The `NetTCPIP` module provides specialized cmdlets for inspecting IP bindings and route tables:

```
# High-level interface overview
Get-NetIPConfiguration

# Detailed inventory of assigned IPv4 addresses
Get-NetIPAddress -AddressFamily IPv4 | Format-Table -Property InterfaceAlias, IPAddress,
PrefixLength, InterfaceIndex

# Active IPv4 routing table
Get-NetRoute -AddressFamily IPv4 | Where-Object DestinationPrefix -ne 'ff00::/8' | Sort-
Object InterfaceIndex
```

Audit remote port reachability and routing paths using `Test-NetConnection`:

```
# Probe gateway or DNS port connectivity
Test-NetConnection -ComputerName 192.0.2.1 -Port 53
```

Expected output:

```
ComputerName     : 192.0.2.1
RemoteAddress    : 192.0.2.1
RemotePort       : 53
InterfaceAlias   : Ethernet0
SourceAddress    : 192.0.2.10
TcpTestSucceeded : True
```

NIC Teaming for High Availability:

If multiple virtual network interfaces on distinct Proxmox bridges are assigned to the VM, combine them into a resilient team without third-party software:

```
# Create switch-independent NIC team with dynamic hashing
New-NetLbfoTeam -Name "Team01" `
  -TeamMembers "Ethernet0", "Ethernet1" `
  -TeamingMode SwitchIndependent `
  -LoadBalancingAlgorithm Dynamic

# Audit team health
Get-NetLbfoTeam | Format-Table -Property Name, Members, TeamingMode, Status
```

Environment Variables and Machine PATH Management:

Modifications to `$env:PATH` within an active PowerShell session persist only for the lifetime of that process. To append custom script paths (such as `C:\Scripts`) globally across all processes and reboots, use the .NET `System.Environment` API:

```
# Retrieve machine-level PATH
$CurrentPath = [System.Environment]::GetEnvironmentVariable("Path", "Machine")

# Append directory if not present
if ($CurrentPath -notlike "*C:\Scripts*") {
    $NewPath = "$CurrentPath;C:\Scripts"
    [System.Environment]::SetEnvironmentVariable("Path", $NewPath, "Machine")
}

# Define persistent machine-level environment variable
[System.Environment]::SetEnvironmentVariable("ADMIN_ENV", "Production", "Machine")
```

Configuring PowerShell Script Execution Policies

By default on Windows Server 2025, the script execution policy is configured to `RemoteSigned` for local server sessions, but can restrict automation scripts deployed via SCP or downloaded from internal repositories. Understanding and configuring the execution policy is essential for seamless script execution on Server Core:

```
# 1. Audit current execution policies across all scopes
Get-ExecutionPolicy -List | Format-Table -Property Scope, ExecutionPolicy

# 2. Enforce RemoteSigned policy at Machine level
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope LocalMachine -Force

# 3. Unblock scripts transferred from external systems (removes Zone.Identifier alternate
data streams)
Unblock-File -Path "C:\Scripts\*.ps1"
```

> **Execution Policies are Safeguards, Not Hard Security Boundaries:** Execution policies are designed to prevent administrators from unintentionally running untrusted scripts, not to defend against malicious authenticated users. Automation tasks launched via scheduled jobs or CI/CD runners should explicitly invoke `-ExecutionPolicy Bypass` within the calling wrapper.

Storage and Partition Management without diskmgmt.msc

In virtualized environments like Proxmox VE, expanding virtual disks or attaching additional storage arrays is a routine operation. On Server Core, disk partitioning is controlled through the `Storage` module.

Inventorying Disks and Partition Layouts:

```
Get-Disk | Format-Table Number, FriendlyName, OperationalStatus, Size, PartitionStyle
```

Output:

```
Number FriendlyName         OperationalStatus      Size PartitionStyle
------ ------------         -----------------      ---- --------------
     0 QEMU HARDDISK        OK                   60 GB  GPT
```

□□ Praktisches Beispiel: Online Live Disk Expansion without Downtime

Expanding a virtual disk on the Proxmox host (e.g., `qm resize 104 scsi0 +20G`) leaves the added capacity as unallocated space at the end of the disk.

Extend the partition and NTFS filesystem live without reboots:

```
# 1. Rescan disk bus to register hypervisor storage expansion
Update-Disk -Number 0

# 2. Query maximum allowable partition size on drive C:
$MaxSize = (Get-PartitionSupportedSize -DiskNumber 0 -PartitionNumber 3).SizeMax

# 3. Expand partition to fill available storage
Resize-Partition -DiskNumber 0 -PartitionNumber 3 -Size $MaxSize
```

Verify the expanded volume:

```
Get-Volume -DriveLetter C | Format-Table DriveLetter, FileSystemLabel, FileSystem, Size,
SizeRemaining
```

Initializing and Formatting Secondary Data Disks:

When attaching an additional virtual disk for application data (e.g., Disk `1`), provision it in a single pipeline:

```
Initialize-Disk -Number 1 -PartitionStyle GPT -PassThru |
    New-Partition -AssignDriveLetter -UseMaximumSize |
    Format-Volume -FileSystem NTFS -NewFileSystemLabel "Data" -Confirm:$false
```

Storage Spaces and Software Mirroring:

If multiple unpartitioned virtual disks are attached (e.g., Disks `1` and `2`), combine them into a software mirror pool:

```
# 1. Identify unpartitioned physical disks eligible for pooling
$Disks = Get-PhysicalDisk -CanPool $true

# 2. Create storage pool
$Pool = New-StoragePool -FriendlyName "DataPool" `
  -StorageSubsystemFriendlyName "Windows Storage*" `
  -PhysicalDisks $Disks

# 3. Provision mirrored virtual disk (RAID-1 equivalent)
$VDisk = New-VirtualDisk -StoragePoolFriendlyName "DataPool" `
```

```
    -FriendlyName "MirrorVolume" `
    -ResiliencySettingName Mirror `
    -UseMaximumSize

# 4. Initialize, partition, and format mirrored disk
$VDisk | Get-Disk | Initialize-Disk -PartitionStyle GPT -PassThru |
    New-Partition -AssignDriveLetter -UseMaximumSize |
    Format-Volume -FileSystem NTFS -NewFileSystemLabel "MirrorData" -Confirm:$false
```

Local Users and Group Management

Server Core lacks the local users and groups snap-in (`lusrmgr.msc`). Manage accounts using `Microsoft.PowerShell.LocalAccounts`:

Auditing Local Accounts:

```
Get-LocalUser | Format-Table Name, Enabled, PasswordRequired, PasswordLastSet, LastLogon
```

Provisioning a Secondary Administrative User:

```
# Prompt for secure credential string
$Password = Read-Host -AsSecureString "Enter password for secondary administrator"

# Provision account
New-LocalUser -Name "secadmin" -Password $Password -FullName "Secondary Admin" -
Description "Emergency Administrator"

# Add account to local Administrators group
Add-LocalGroupMember -Group "Administrators" -Member "secadmin"
```

Controlling Account Policies:

```
# Disable password expiration for dedicated service accounts
Set-LocalUser -Name "secadmin" -PasswordNeverExpires $true

# Unlock accounts following brute-force lockouts
Unlock-LocalUser -Name "secadmin"
```

Local Security Policies without secpol.msc (secedit)

Without `secpol.msc`, configure account lockout thresholds and password complexity using `secedit.exe`:

```
# 1. Export current security database to intermediate template
secedit /export /cfg C:\secpol.inf
```

Modify parameters via PowerShell text replacements:

```
(Get-Content C:\secpol.inf) `
    -replace '^LockoutBadCount\s*=.*', 'LockoutBadCount = 5' `
    -replace '^ResetLockoutCount\s*=.*', 'ResetLockoutCount = 30' `
    -replace '^LockoutDuration\s*=.*', 'LockoutDuration = 30' `
    -replace '^PasswordComplexity\s*=.*', 'PasswordComplexity = 1' |
    Set-Content C:\secpol.inf -Encoding Ascii
```

Commit changes to the security store:

```
# 2. Write configuration into local security database
secedit /configure /db C:\Windows\security\local.sdb /cfg C:\secpol.inf /areas
SECURITYPOLICY

# 3. Refresh group policies immediately and remove temporary artifact
gpupdate /force
del C:\secpol.inf
```

NTP and Time Synchronization

Virtual machine system clocks can drift during hypervisor overcommit or migrations. Accurate timekeeping is critical for Kerberos ticket validation and log correlation.

Audit and configure the Windows Time service using `w32tm.exe`:

Query Active Time Status:

```
w32tm /query /status
```

Output:

```
Leap Indicator: 0(no warning)
Stratum: 2 (secondary reference - syncd by (S)NTP)
Precision: -23 (119.209ns per tick)
Root Delay: 0.0152588s
Root Dispersion: 0.0315482s
ReferenceId: 0xC0000201 (source IP:  192.0.2.1)
```

```
Last Successful Sync Time: 10.11.2026 10:15:32
Source: 192.0.2.1,0x8
Poll Interval: 6 (64s)
```

Configure External NTP Peers:

```
w32tm /config /manualpeerlist:"0.pool.ntp.org,0x8 1.pool.ntp.org,0x8"
/syncfromflags:manual /reliable:yes /update
w32tm /resync /rediscover
```

Controlling Services and Processes

Manage background services via `Microsoft.PowerShell.Management`:

```
# Audit automatic services currently stopped
Get-Service | Where-Object { $_.StartType -eq 'Automatic' -and $_.Status -eq 'Stopped' } |
Format-Table Name, DisplayName

# Restart daemon
Restart-Service -Name sshd -Force

# Change service startup type
Set-Service -Name w32time -StartupType Automatic
Start-Service -Name w32time
```

Process Auditing and Resource Triage:

```
# Identify top 10 memory-consuming processes
Get-Process | Sort-Object WorkingSet64 -Descending | Select-Object -First 10 -Property Id,
ProcessName, CPU, @{Name='WorkingSet_MB'; Expression={[math]::Round($_.WorkingSet64 / 1MB,
2)}} | Format-Table -AutoSize
```

Output:

```
  Id ProcessName  CPU WorkingSet_MB
  -- -----------  --- -------------
2148 svchost    12.45        184.32
1604 sshd        1.12         42.18
2892 powershell  4.88         98.64
 824 lsass       3.21         36.50
```

Terminate unresponsive processes cleanly by PID:

```
Stop-Process -Id 4328 -Force
```

Managing Roles and Features

Manage server roles and features using the `ServerManager` module:

```
# Audit installed roles
Get-WindowsFeature | Where-Object Installed -eq $true | Format-Table Name, DisplayName

# Install DNS Server role including management tools
Install-WindowsFeature -Name DNS -IncludeManagementTools

# Verify installation status
Get-WindowsFeature -Name DNS
```

> □□ **WindowsFeatures vs. WindowsCapabilities:** Full infrastructure roles (such as DNS, IIS, or Active Directory) are installed via `Install-WindowsFeature`. Modular OS components (like OpenSSH Server) are managed as operating system capabilities via `Add-WindowsCapability`.

Active Directory Domain Join via PowerShell:

```
# 1. Test DNS SRV record resolution to domain controller
Resolve-DnsName -Name "corp.example.com"

# 2. Join domain using administrative credentials
$DomainCred = Get-Credential
Add-Computer -DomainName "corp.example.com" -Credential $DomainCred -Restart -Force
```

Authoritative DNS Server Configuration:

Provision forward-lookup zones and records on Server Core:

```
# 1. Create primary forward-lookup zone
Add-DnsServerPrimaryZone -Name "lab.internal" -ZoneFile "lab.internal.dns"

# 2. Add static A and CNAME records
Add-DnsServerResourceRecordA -ZoneName "lab.internal" -Name "gateway" -IPv4Address
"192.0.2.1"
Add-DnsServerResourceRecordA -ZoneName "lab.internal" -Name "pve-node01" -IPv4Address
"192.0.2.2"
Add-DnsServerResourceRecordCName -ZoneName "lab.internal" -Name "dns" -HostNameAlias
"gateway.lab.internal."

# 3. Configure upstream DNS forwarders
Set-DnsServerForwarder -IPAddress "1.1.1.1", "8.8.8.8"

# 4. Audit records
Get-DnsServerResourceRecord -ZoneName "lab.internal" | Format-Table -Property HostName,
RecordType, RecordData
```

Advanced DNS Record Management: Reverse PTR, MX, and TXT Records:

Enterprise DNS zones require reverse lookup zones for IP-to-hostname resolution, alongside specialized mail exchanger (MX) and SPF text records:

```
# 1. Create IPv4 reverse lookup zone for 192.0.2.0/24
Add-DnsServerPrimaryZone -NetworkId "192.0.2.0/24" -ZoneFile "2.0.192.in-addr.arpa.dns"

# 2. Register PTR pointer record for host
Add-DnsServerResourceRecordPtr -ZoneName "2.0.192.in-addr.arpa" -Name "10" -PtrDomainName
"win-core01.lab.internal."

# 3. Add Mail Exchanger (MX) and TXT SPF validation records
Add-DnsServerResourceRecordMX -ZoneName "lab.internal" -Name "@" -MailExchange
"mail.lab.internal." -Preference 10
Add-DnsServerResourceRecord -ZoneName "lab.internal" -Txt -Name "@" -DescriptiveText
```

```
"v=spf1 mx ip4:192.0.2.10 ~all"

# 4. Verify reverse zone entries
Get-DnsServerResourceRecord -ZoneName "2.0.192.in-addr.arpa" | Format-Table HostName,
RecordType, RecordData
```

Active Directory Administration via the ActiveDirectory PowerShell Module:

Once joined to an Active Directory domain, the native `ActiveDirectory` module enables headless domain operations directly over your SSH terminal session:

```
# Import the Active Directory administrative module
Import-Module ActiveDirectory

# Discover active domain controllers and forest topology
Get-ADDomainController -Discover
Get-ADDomain | Format-Table -Property Name, DomainMode, Forest

# Query active administrative accounts within the domain
Get-ADUser -Filter 'Enabled -eq $true' -SearchBase "CN=Users,DC=corp,DC=example,DC=com" |
    Format-Table -Property SamAccountName, Name, UserPrincipalName

# Audit domain group memberships
Get-ADGroupMember -Identity "Domain Admins" | Select-Object -Property SamAccountName, Name
```

File Server and SMB Shares without GUI:

```
Install-WindowsFeature -Name FS-FileServer
New-Item -Path "D:\Shares\Deploy" -ItemType Directory -Force | Out-Null

New-SmbShare -Name "Deploy" `
```

```
    -Path "D:\Shares\Deploy" `
    -FullAccess "BUILTIN\Administrators" `
    -ChangeAccess "NT AUTHORITY\Authenticated Users" `
    -Description "Central software deployment share"

Get-SmbShare | Format-Table -Property Name, Path, Description
Get-SmbSession | Format-Table -Property SessionId, ClientComputerName, ClientUserName,
NumOpens
```

NTFS File System ACLs without Explorer Security Tab:

Manage file system permissions through .NET `System.Security.AccessControl`:

```
$Path = "D:\Shares\Deploy"
$Acl = Get-Acl -Path $Path

# Block inheritance and convert existing permissions to explicit entries
$Acl.SetAccessRuleProtection($true, $true)

# Grant Full Control to Administrators
$AdminRule = New-Object System.Security.AccessControl.FileSystemAccessRule(
    "BUILTIN\Administrators",
    "FullControl",
    "ContainerInherit,ObjectInherit",
    "None",
    "Allow"
)
$Acl.AddAccessRule($AdminRule)

# Grant Read & Execute to Authenticated Users
$UserRule = New-Object System.Security.AccessControl.FileSystemAccessRule(
    "NT AUTHORITY\Authenticated Users",
    "ReadAndExecute",
    "ContainerInherit,ObjectInherit",
    "None",
    "Allow"
)
```

```
$Acl.AddAccessRule($UserRule)

Set-Acl -Path $Path -AclObject $Acl
(Get-Acl -Path $Path).Access | Format-Table -Property IdentityReference, FileSystemRights,
AccessControlType, IsInherited
```

Certificate Store Management on Server Core without certmgr.msc

Manage X.509 certificates via the `Cert:` PSDrive:

```
# Query machine certificate store
Get-ChildItem -Path "Cert:\LocalMachine\My" | Format-Table -Property Subject, Thumbprint,
NotAfter

# Generate self-signed certificate for internal services
$Cert = New-SelfSignedCertificate `
  -DnsName "win-core01.lab.internal" `
  -CertStoreLocation "Cert:\LocalMachine\My" `
  -NotAfter (Get-Date).AddYears(2) `
  -KeyLength 2048 `
  -FriendlyName "WIN-CORE01 Test Certificate"

# Import enterprise CA root certificate
Import-Certificate -FilePath "C:\Certs\InternalRootCA.crt" -CertStoreLocation
"Cert:\LocalMachine\Root"

# Audit certificates expiring within 30 days
Get-ChildItem -Path "Cert:\LocalMachine\My" |
    Where-Object { $_.NotAfter -lt (Get-Date).AddDays(30) } |
    Format-Table -Property Subject, NotAfter, Thumbprint
```

Windows Event Log Auditing via Get-WinEvent

Without `eventvwr.msc`, `Get-WinEvent` serves as the primary tool for diagnostic auditing.

> ⚡ **Performance Rule for Event Logs:** Never fetch large event logs unconstrained and filter them downstream using `Where-Object`. Always pass a `-FilterHashtable` so filtering occurs directly within the Windows kernel event engine.

🖥️ Praktisches Beispiel: Querying System Errors and Warnings

```
$StartTime = (Get-Date).AddDays(-1)
$Filter = @{
    LogName   = 'System'
    Level     = 2, 3       # 2 = Error, 3 = Warning
    StartTime = $StartTime
}

Get-WinEvent -FilterHashtable $Filter -MaxEvents 50 | Format-Table TimeCreated, Id,
ProviderName, Message -AutoSize
```

🖥️ Praktisches Beispiel: Isolating Failed Logon Attempts (Event ID 4625)

```
$SecurityFilter = @{
    LogName = 'Security'
    Id      = 4625
}

Get-WinEvent -FilterHashtable $SecurityFilter -MaxEvents 10 | Select-Object TimeCreated,
Id, Message | Format-List
```

Deploying Web Server (IIS) on Server Core

Deploying IIS on Server Core is a model deployment pattern: idle memory consumption is less than 80 MiB.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

```
# 1. Install IIS and management modules
Install-WindowsFeature -Name Web-Server, Web-Mgmt-Tools
Import-Module WebAdministration

# 2. Prepare document directory and index page
$WebRoot = "C:\inetpub\docs"
New-Item -Path $WebRoot -ItemType Directory -Force | Out-Null
Set-Content -Path "$WebRoot\index.html" -Value "<h1>AdminDocs Windows Server 2025
Core</h1><p>Status: Online</p>"

# 3. Create isolated Application Pool and bind Website
New-WebAppPool -Name "DocsPool"
Set-ItemProperty -Path "IIS:\AppPools\DocsPool" -Name "managedRuntimeVersion" -Value ""
New-WebSite -Name "DocsApp" -Port 80 -PhysicalPath $WebRoot -ApplicationPool "DocsPool" -
Force

# 4. Open HTTP and HTTPS firewall ports
New-NetFirewallRule -Name "IIS-HTTP-In" -DisplayName "Webserver HTTP (Port 80)" -Direction
Inbound -Protocol TCP -LocalPort 80 -Action Allow
New-NetFirewallRule -Name "IIS-HTTPS-In" -DisplayName "Webserver HTTPS (Port 443)" -
Direction Inbound -Protocol TCP -LocalPort 443 -Action Allow
```

Headless Network Packet Traces without Wireshark (netsh trace)

Diagnose complex network, SMB, or Kerberos issues at the packet layer using built-in `netsh trace`:

□□ Praktisches Beispiel: Circular Buffer Network Packet Capture

```
# Start packet trace into 256 MB circular ring buffer
netsh trace start capture=yes tracefile=C:\trace.etl maxsize=256 filemode=circular
overwrite=yes
```

```
# Reproduce the network error, then stop tracing
netsh trace stop
```

Transfer the ETL trace to your Linux workstation:

```
scp -i ~/.ssh/id_ed25519 Administrator@192.0.2.10:C:/trace.etl ~/trace.etl
```

Convert the trace to PCAPNG format using `etl2pcapng` and analyze in Wireshark:

```
etl2pcapng ~/trace.etl ~/trace.pcapng
wireshark ~/trace.pcapng
```

Updates, Maintenance and Baseline Hardening

Following initial provisioning, focus transitions to sustained operational reliability. On a headless server, patch management, endpoint security, firewall segmentation, storage reclaiming, and backup consistency require automated, auditable procedures.

Patch Management and Windows Update

Windows Server Core lacks a graphical Windows Update dialog. Two proven operational workflows exist for applying security updates:

Interactive Patching via SConfig:

For ad-hoc maintenance sessions at the console, Option `6) Download and Install Updates` in SConfig provides the quickest path:

```
sconfig
```

Select Option 6 to choose between querying only recommended patches or all available packages. SConfig lists identified KB packages and allows targeted or comprehensive installation accompanied by automated reboots.

Scripted Patching via PSWindowsUpdate:

In standardized environments and automated maintenance windows, manage updates programmatically using the `PSWindowsUpdate` PowerShell module:

```
# Install NuGet package provider and PSWindowsUpdate module
Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force
Install-Module -Name PSWindowsUpdate -Force
```

Control the entire update lifecycle via script:

```
# Query pending updates without installing
Get-WindowsUpdate

# Download, install all updates, and reboot if required
Install-WindowsUpdate -MicrosoftUpdate -AcceptAll -AutoReboot
```

> ⚠ **Plan Maintenance Windows Carefully:** The `-AutoReboot` flag triggers an immediate reboot whenever a patch requires it. Execute this command in production solely within authorized maintenance windows.

Verifying Installed Updates and Hotfix History:

Audit installed security updates using `Get-HotFix`:

```
Get-HotFix | Sort-Object InstalledOn -Descending | Format-Table -Property HotFixID,
Description, InstalledBy, InstalledOn
```

The actual updates and timestamps returned depend on the system release state and patch level. The schematic example below illustrates the tabular output format:

```
HotFixID   Description     InstalledBy          InstalledOn
--------   -----------     -----------          -----------
KBxxxxxxx  Security Update NT AUTHORITY\SYSTEM  ...
KBxxxxxxx  Update          NT AUTHORITY\SYSTEM  ...
```

Microsoft Defender Antivirus on Server Core

Microsoft Defender Antivirus is active and integrated out-of-the-box on Windows Server 2025 Core. Manage endpoint protection using the `Defender` module:

Audit Protection State:

```
Get-MpComputerStatus | Select-Object -Property AntivirusEnabled,
RealTimeProtectionEnabled, AntivirusSignatureVersion, AntivirusSignatureLastUpdated
```

Update Signatures and Run Scans:

```
# Force immediate signature definition update
Update-MpSignature

# Configure signature fallback to Microsoft Malware Protection Center if WSUS is
unreachable
Set-MpPreference -SignatureFallbackOrder "MicrosoftUpdateServer|MMPC"

# Audit antimalware engine version and active signatures
Get-MpComputerStatus | Select-Object -Property AMEngineVersion, AntivirusSignatureVersion,
NISSignatureVersion

# Initiate background quick scan
Start-MpScan -ScanType QuickScan
```

Configure Path and Extension Exclusions:

Scanning high-throughput database paths or virtual disks severely degrades I/O latency. Define exclusions via PowerShell:

```
# Exclude database storage paths
Add-MpPreference -ExclusionPath "C:\Data\Database"

# Exclude virtual hard disk extensions
```

```
Add-MpPreference -ExclusionExtension ".vhdx"

# Audit configured exclusions
Get-MpPreference | Select-Object -ExpandProperty ExclusionPath
```

Inspect Threat Detections:

```
Get-MpThreatDetection | Select-Object -Property InitialDetectionTime, ThreatName,
Resources
```

Hardening Windows Firewall and Restricting Access

Windows Defender Firewall is active across all profiles (*Domain*, *Private*, *Public*) by default, blocking unsolicited inbound packets.

Audit Profile Status:

```
Get-NetFirewallProfile | Format-Table -Property Name, Enabled, DefaultInboundAction,
DefaultOutboundAction
```

In hardened environments, `DefaultInboundAction` should always report `Block`.

Restrict OpenSSH Inbound Port to Management Subnets:

By default, the OpenSSH firewall rule allows connections from any source address. Restrict management traffic to your administrative subnet:

```
# Restrict SSH ingress strictly to 192.0.2.0/24
Set-NetFirewallRule -Name 'OpenSSH-Server-In-TCP' -RemoteAddress '192.0.2.0/24'

# Verify rule binding
Get-NetFirewallRule -Name 'OpenSSH-Server-In-TCP' | Get-NetFirewallAddressFilter | Format-List RemoteAddress
```

Enable Dropped Packet Logging:

```
Set-NetFirewallProfile -Profile Domain, Private -LogBlocked True -LogFileName "$env:SystemRoot\System32\LogFiles\Firewall\pfirewall.log"
```

System Baseline Hardening against CIS Security Standards

Center for Internet Security (CIS) benchmarks recommend disabling legacy name resolution and authentication protocols that are vulnerable to credential interception:

1. Disable LLMNR (Link-Local Multicast Name Resolution):

LLMNR broadcasts multicast queries when DNS fails, leaving systems vulnerable to rogue LLMNR poisoning. Disable it via Registry:

```
$DnsPolicyPath = "HKLM:\SOFTWARE\Policies\Microsoft\Windows NT\DNSClient"
if (-not (Test-Path $DnsPolicyPath)) { New-Item -Path $DnsPolicyPath -Force | Out-Null }
Set-ItemProperty -Path $DnsPolicyPath -Name "EnableMulticast" -Value 0 -Type DWord
```

2. Disable NetBIOS over TCP/IP across All Interfaces:

```
$Adapters = Get-CimInstance -ClassName Win32_NetworkAdapterConfiguration | Where-Object
IPEnabled
foreach ($adapter in $Adapters) {
    Invoke-CimMethod -InputObject $adapter -MethodName SetTcpipNetbios -Arguments
@{TcpipNetbiosOptions = 2} | Out-Null
}
```

3. Restrict NTLM Authentication and Enforce NTLMv2:

```
Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\Lsa" `
  -Name "LmCompatibilityLevel" -Value 5 -Type DWord
```

4. Block WDigest Plaintext Credential Caching in LSASS Memory:

```
Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest"
`
  -Name "UseLogonCredential" -Value 0 -Type DWord
```

QEMU Guest Agent and VSS Snapshots for Proxmox VE

Deploying the **QEMU Guest Agent** (`qemu-ga`) on Windows Server Core is mandatory for application-consistent backups on Proxmox VE 9.

Significance for Backup Consistency:

Without the guest agent, hypervisor snapshots are merely *crash-consistent*. The QEMU Guest Agent integrates into the Windows Volume Shadow Copy Service (VSS). When a backup triggers via Proxmox Backup Server, the agent commands VSS to freeze file systems and flush database buffers (*fsfreeze*) for a few milliseconds, guaranteeing an *application-consistent* backup state.

□□ Praktisches Beispiel: Installing QEMU Guest Agent on Server Core

With `virtio-win.iso` attached to drive `D:`, install the agent silently:

```
# Execute silent MSI installation
Start-Process -FilePath "msiexec.exe" -ArgumentList "/i D:\guest-agent\qemu-ga-x86_64.msi
/quiet /norestart" -Wait

# Configure service for automatic launch and start
Set-Service -Name "QEMU-GA" -StartupType Automatic
Start-Service -Name "QEMU-GA"

# Verify service state
Get-Service -Name "QEMU-GA" | Format-Table Name, Status, StartType
```

Enable the agent on the Proxmox host:

```
qm set 104 -agent 1
```

Verify guest agent responsiveness directly from the Proxmox host shell:

```
# Ping the guest agent to confirm socket communication
qm guest cmd 104 ping

# Query network interfaces reported by the Windows guest agent
qm guest cmd 104 network-get-interfaces

# Retrieve guest operating system information
qm guest cmd 104 get-osinfo
```

Proxmox VE immediately reports accurate IP addresses, filesystem mountpoints, and guest status in the dashboard.

Verifying VSS Writer Health:

Audit registered VSS writers using `vssadmin`:

```
vssadmin list writers
```

All writers must report state `[1] Stable` without errors:

```
Writer name: 'System Writer'
   Writer Id: {e81062d3-1f76-4d37-a4b7-8331db428a5e}
   Writer Instance Id: {4a5b6c7d-8e9f-0a1b-2c3d-4e5f6a7b8c9d}
   State: [1] Stable
   Last error: No error
```

Optimizing Thin-Provisioning and Trim/Discard

When running virtual machines on ZFS, Ceph, or LVM-Thin, disk images grow dynamically. When files are deleted inside Windows, the hypervisor remains unaware by default, leaving physical blocks allocated.

Ensure `discard=on` is configured on the SCSI drive in Proxmox, then trigger manual TRIM/UNMAP:

```
Optimize-Volume -DriveLetter C -Defrag -Verbose
```

Output:

```
VERBOSE: Invoking slab consolidation on Windows (C:)...
VERBOSE: Invoking re-trim on Windows (C:)...
VERBOSE: Re-Trim: 100% complete.
```

Scheduled Tasks and Automated Maintenance without taskschd.msc

Manage periodic maintenance routines using the `ScheduledTasks` module:

□□ Praktisches Beispiel: Maintenance Script Maintain-ServerCore.ps1

Create `C:\Scripts\Maintain-ServerCore.ps1`:

```
# Maintain-ServerCore.ps1
```

```
# Automated maintenance routine for Windows Server 2025 Core

$LogPath = "C:\Scripts\Logs"
if (-not (Test-Path $LogPath)) { New-Item -Path $LogPath -ItemType Directory -Force | Out-
Null }
$LogFile = "$LogPath\Maintenance_$(Get-Date -Format 'yyyy-MM-dd').log"

function Write-Log {
    param ([string]$Message)
    $Timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    "[$Timestamp] $Message" | Tee-Object -FilePath $LogFile -Append
}

Write-Log "=== Starting Server Core Maintenance Routine ==="

# 1. Update Defender signatures and execute quick scan
Write-Log "[1/4] Updating Microsoft Defender signatures..."
Update-MpSignature
Write-Log "[1/4] Running QuickScan..."
Start-MpScan -ScanType QuickScan

# 2. Disk optimization and hypervisor TRIM
Write-Log "[2/4] Executing TRIM/UNMAP on system volume C:..."
Optimize-Volume -DriveLetter C -Defrag -Verbose 4>&1 | ForEach-Object { Write-Log "  $_" }

# 3. Clean temporary system files older than 7 days
Write-Log "[3/4] Purging stale temporary files..."
$TempDirs = @("$env:SystemRoot\Temp", "$env:TEMP")
$CutOffDate = (Get-Date).AddDays(-7)

foreach ($dir in $TempDirs) {
    if (Test-Path $dir) {
        Get-ChildItem -Path $dir -Recurse -File -ErrorAction SilentlyContinue |
            Where-Object { $_.LastWriteTime -lt $CutOffDate } |
            Remove-Item -Force -ErrorAction SilentlyContinue
    }
}

# 4. Validate VSS writer status
Write-Log "[4/4] Validating Volume Shadow Copy Writers..."
$FailedWriters = vssadmin list writers | Select-String -Pattern "State: \[[^1]\]"
if ($FailedWriters) {
    Write-Log "WARNING: One or more VSS writers are in a failed state!"
} else {
```

```
    Write-Log "All VSS writers are in state [1] Stable."
}

Write-Log "=== Maintenance routine completed successfully ==="
```

Register Weekly Scheduled Task:

```
$Action = New-ScheduledTaskAction -Execute "powershell.exe" `
  -Argument "-NoProfile -ExecutionPolicy Bypass -File C:\Scripts\Maintain-ServerCore.ps1"

$Trigger = New-ScheduledTaskTrigger -Weekly -DaysOfWeek Sunday -At 03:00
$Principal = New-ScheduledTaskPrincipal -UserId "NT AUTHORITY\SYSTEM" -LogonType
ServiceAccount -RunLevel Highest

Register-ScheduledTask -TaskName "ServerCore-Maintenance" `
  -Action $Action `
  -Trigger $Trigger `
  -Principal $Principal `
  -Description "Weekly maintenance routine (Defender, TRIM, VSS, Temp)"

Get-ScheduledTask -TaskName "ServerCore-Maintenance"
```

Performance Monitoring and Capacity Planning with Get-Counter

Query real-time performance metrics via `Get-Counter`:

```
$Counters = @(
    "\Processor(_Total)\% Processor Time",
    "\Memory\Available MBytes",
    "\PhysicalDisk(_Total)\% Disk Time",
    "\PhysicalDisk(_Total)\Current Disk Queue Length",
```

```
    "\Network Interface(*)\Bytes Total/sec"
)

Get-Counter -Counter $Counters | Select-Object -ExpandProperty CounterSamples |
    Format-Table -Property Path, CookedValue -AutoSize
```

Output:

```
Path                                                    CookedValue
----                                                    -----------
\\win-core01\processor(_total)\% processor time                3.12
\\win-core01\memory\available mbytes                       6842.00
\\win-core01\physicaldisk(_total)\% disk time                 0.45
\\win-core01\physicaldisk(_total)\current disk queue length   0.00
\\win-core01\network interface(ethernet0)\bytes total/sec     1248.50
```

Log metrics to CSV for offline analysis:

```
Get-Counter -Counter "\Processor(_Total)\% Processor Time", "\Memory\Available MBytes" `
  -SampleInterval 5 -MaxSamples 60 |
  Export-Counter -Path "C:\Scripts\Logs\Performance_$(Get-Date -Format 'yyyy-MM-
dd_HHmm').csv" -FileFormat CSV
```

Central WSUS and Group Policy Management without gpedit.msc

Manage update policies directly in the registry:

```
$WsusPath = "HKLM:\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate"
$AuPath   = "$WsusPath\AU"

if (-not (Test-Path $WsusPath)) { New-Item -Path $WsusPath -Force | Out-Null }
if (-not (Test-Path $AuPath))   { New-Item -Path $AuPath -Force | Out-Null }

Set-ItemProperty -Path $WsusPath -Name "WUServer" -Value
"http://wsus.corp.example.com:8530"
Set-ItemProperty -Path $WsusPath -Name "WUStatusServer" -Value
"http://wsus.corp.example.com:8530"

# 4 = Automatically download and schedule installation
Set-ItemProperty -Path $AuPath -Name "AUOptions" -Value 4 -Type DWord
Set-ItemProperty -Path $AuPath -Name "ScheduledInstallDay" -Value 7 -Type DWord
Set-ItemProperty -Path $AuPath -Name "ScheduledInstallTime" -Value 4 -Type DWord

Restart-Service -Name wuauserv
```

WinSxS Component Store Cleanup with DISM

Reclaim disk space in the `WinSxS` component store using DISM:

```
# Analyze cleanup eligibility
dism.exe /Online /Cleanup-Image /AnalyzeComponentStore

# Reclaim superseded updates
dism.exe /Online /Cleanup-Image /StartComponentCleanup

# Reset base state to permanently reclaim 3-6 GiB
dism.exe /Online /Cleanup-Image /StartComponentCleanup /ResetBase
```

> **Implication of /ResetBase:** Executing `/ResetBase` prevents uninstalling currently active cumulative patches. Apply this command only after patches have proven stable in production for several days.

System File Verification and Image Repair with SFC and DISM

```
# Scan protected operating system files
sfc.exe /scannow

# Analyze and repair component store online via Windows Update
dism.exe /Online /Cleanup-Image /ScanHealth
dism.exe /Online /Cleanup-Image /CheckHealth
dism.exe /Online /Cleanup-Image /RestoreHealth

# Offline repair targeting mounted ISO WIM image
dism.exe /Online /Cleanup-Image /RestoreHealth /Source:wim:D:\sources\install.wim:1
/LimitAccess
```

Storage Directory Usage Audit in PowerShell:

```
Get-ChildItem -Path "C:\" -Directory -Force | ForEach-Object {
    $DirName = $_.Name
    $Size = (Get-ChildItem -Path $_.FullName -Recurse -File -Force -ErrorAction
SilentlyContinue |
             Measure-Object -Property Length -Sum).Sum
    [PSCustomObject]@{
        Directory = $DirName
        Size_GB   = [math]::Round($Size / 1GB, 2)
    }
} | Sort-Object Size_GB -Descending | Format-Table -AutoSize
```

BitLocker Drive Encryption Management without GUI

In enterprise datacenter environments, encrypting the virtual operating system disk protects data at rest against unauthorized extraction of VM disk images from shared SAN or Ceph pools. On Server Core, manage BitLocker via PowerShell and `manage-bde.exe`:

```
# 1. Audit BitLocker protection state across all volumes
Get-BitLockerVolume | Format-Table -Property MountPoint, VolumeType, VolumeStatus,
EncryptionPercentage, KeyProtector

# 2. Enable BitLocker on the operating system drive utilizing the virtual TPM
Enable-BitLocker -MountPoint "C:" `
  -EncryptionMethod XtsAes256 `
  -UsedSpaceOnly `
  -TpmProtector

# 3. Add numerical recovery password protector for emergency disaster recovery
Add-BitLockerKeyProtector -MountPoint "C:" -RecoveryPasswordProtector

# 4. Display and backup the generated 48-digit numerical recovery password
(Get-BitLockerVolume -MountPoint "C:").KeyProtector |
    Where-Object { $_.KeyProtectorType -eq 'RecoveryPassword' } |
    Select-Object -ExpandProperty RecoveryPassword
```

> ⚠ **Backup Recovery Password Immediately:** Store the generated 48-digit numerical recovery key securely outside the virtual machine. If the virtual TPM NVRAM disk is recreated or desynchronized during hypervisor migration, the recovery key is the only method to unlock the volume.

Automated Windows Event Log Archiving and Compression

Operational compliance mandates periodic archival of event logs before circular buffers overwrite security events. On Server Core, automate log archival using `wevtutil.exe` and PowerShell:

□□ Praktisches Beispiel: Automated Log Archival Script Archive-EventLogs.ps1

Create `C:\Scripts\Archive-EventLogs.ps1`:

```
# Archive-EventLogs.ps1
# Automated monthly export and compression of operational logs

$ArchiveBase = "C:\Scripts\LogArchive"
if (-not (Test-Path $ArchiveBase)) { New-Item -Path $ArchiveBase -ItemType Directory -
Force | Out-Null }
$Timestamp = Get-Date -Format "yyyyMMdd"

# 1. Export critical event channels to raw EVTX files
wevtutil.exe epl System "$ArchiveBase\System_$Timestamp.evtx"
wevtutil.exe epl Security "$ArchiveBase\Security_$Timestamp.evtx"
wevtutil.exe epl "OpenSSH/Operational" "$ArchiveBase\OpenSSH_$Timestamp.evtx"

# 2. Compress exported EVTX files into ZIP archive and remove raw artifacts
Compress-Archive -Path "$ArchiveBase\*_$Timestamp.evtx" -DestinationPath
"$ArchiveBase\EventLogs_$Timestamp.zip" -Force
Remove-Item "$ArchiveBase\*_$Timestamp.evtx" -Force

# 3. Purge archives older than 90 days
Get-ChildItem -Path $ArchiveBase -Filter "*.zip" |
    Where-Object { $_.LastWriteTime -lt (Get-Date).AddDays(-90) } |
    Remove-Item -Force
```

Comprehensive Health Audit Script: Test-ServerHealth.ps1

Before concluding maintenance windows or handing systems over to application teams, run an automated verification audit covering compute resources, disk thresholds, core background daemons, and VSS writer health:

□□ Praktisches Beispiel: System Health Check Script Test-ServerHealth.ps1

Create and execute `C:\Scripts\Test-ServerHealth.ps1`:

```
# Test-ServerHealth.ps1
# Comprehensive operational health check for Windows Server Core

Write-Host "=== System Health Audit for $env:COMPUTERNAME ===" -ForegroundColor Cyan

# 1. CPU & Memory Check
$OS = Get-CimInstance Win32_OperatingSystem
$FreeRAM_GB = [math]::Round($OS.FreePhysicalMemory / 1MB, 2)
$TotalRAM_GB = [math]::Round($OS.TotalVisibleMemorySize / 1MB, 2)
Write-Host "[1/5] Memory: $FreeRAM_GB GB free of $TotalRAM_GB GB" -ForegroundColor Green

# 2. Disk Space Check
Get-Volume | Where-Object DriveLetter -ne $null | ForEach-Object {
    $FreePct = [math]::Round(($_.SizeRemaining / $_.Size) * 100, 1)
    Write-Host "[2/5] Drive $($_.DriveLetter): $FreePct% free
($([math]::Round($_.SizeRemaining/1GB,1)) GB)" -ForegroundColor Green
}

# 3. Critical Service Status
$CriticalServices = @("sshd", "w32time", "wuauserv", "QEMU-GA", "EventLog")
foreach ($svc in $CriticalServices) {
    $Status = (Get-Service -Name $svc -ErrorAction SilentlyContinue).Status
    $Color = $Status -eq 'Running' ? 'Green' : 'Red'
    Write-Host "[3/5] Service $svc: $Status" -ForegroundColor $Color
}

# 4. VSS Writers Validation
$VssErrors = vssadmin list writers | Select-String -Pattern "State: \[[^1]\]"
if (-not $VssErrors) {
    Write-Host "[4/5] VSS Writers: All registered writers Stable" -ForegroundColor Green
} else {
    Write-Host "[4/5] VSS Writers: ERRORS DETECTED" -ForegroundColor Red
}

# 5. Defender Status
$Defender = Get-MpComputerStatus
Write-Host "[5/5] Defender RealTimeProtection: $($Defender.RealTimeProtectionEnabled)" -
ForegroundColor Green
Write-Host "=== Health Audit Completed Successfully ===" -ForegroundColor Cyan
```

Controlled Maintenance Reboots and Shutdowns

```
# Controlled immediate reboot
Restart-Computer -Force

# Timed reboot notifying active terminal sessions
shutdown.exe /r /t 60 /c "Scheduled maintenance reboot following update cycle"

# Controlled power down
Stop-Computer -Force
```

Command Reference for Daily Administration

For rapid operational reference, the cheatsheet below aggregates the primary commands for Windows Server Core. All commands execute directly within an SSH terminal or the local PowerShell console:

Launch SConfig Launches the menu-driven server configuration utility for hostnames, domains, networking, and updates

$ `sconfig`

Disable SConfig Autolaunch Prevents SConfig from launching automatically upon interactive Administrator logon

$ `Set-SConfig -AutoLaunch $false`

Display Network Interfaces and IP Addresses Lists all physical and virtual network adapters along with assigned IP configurations

$ `Get-NetIPConfiguration`

Configure Static IPv4 Address Assigns a static IP address, subnet prefix, and default gateway to the primary network adapter

```
$ New-NetIPAddress -InterfaceAlias "Ethernet0" -IPAddress 192.0.2.10 -PrefixLength 24 -DefaultGateway 192.0.2.1
```

Configure DNS Resolvers Defines primary and secondary DNS resolvers for network name resolution

```
$ Set-DnsClientServerAddress -InterfaceAlias "Ethernet0" -ServerAddresses ("192.0.2.1", "1.1.1.1")
```

Test Remote Port Reachability Verifies TCP network reachability to remote services such as DNS or default gateways

```
$ Test-NetConnection -ComputerName 192.0.2.1 -Port 53
```

Enable and Start OpenSSH Daemon Configures the OpenSSH service for automatic startup and launches the daemon

```
$ Set-Service -Name sshd -StartupType Automatic; Start-Service -Name sshd
```

Set PowerShell as Default SSH Shell Configures native Windows PowerShell as the default login shell for inbound SSH sessions

```
$ New-ItemProperty -Path "HKLM:\SOFTWARE\OpenSSH" -Name DefaultShell -Value "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -PropertyType String -Force
```

Enforce ACLs on administrators_authorized_keys Strips inherited permissions and restricts access strictly to SYSTEM and Administrators

```
$ icacls.exe $env:ProgramData\ssh\administrators_authorized_keys /inheritance:r /grant "NT AUTHORITY\SYSTEM:(F)" /grant "BUILTIN\Administrators:(F)"
```

Restrict SSH Firewall Rule to Management Subnet Limits inbound SSH traffic strictly to an authorized administrative subnet

```
$ Set-NetFirewallRule -Name "OpenSSH-Server-In-TCP" -RemoteAddress "192.0.2.0/24"
```

Inventory Detected Storage Disks Displays physical and virtual storage disks along with partition styles and capacities

```
$ Get-Disk | Format-Table Number, FriendlyName, OperationalStatus, Size, PartitionStyle
```

Online Live Expansion of System Drive C: Rescans storage bus and expands partition 3 to utilize hypervisor-allocated disk space

```
$ Update-Disk -Number 0; Resize-Partition -DiskNumber 0 -PartitionNumber 3 -Size ((Get-PartitionSupportedSize -DiskNumber 0 -PartitionNumber 3).SizeMax)
```

Initialize and Format Secondary Disk Initializes an unpartitioned drive with GPT and formats an NTFS data volume in one pipeline

```
$ Initialize-Disk -Number 1 -PartitionStyle GPT -PassThru | New-Partition -AssignDriveLetter -UseMaximumSize | Format-Volume -FileSystem NTFS -NewFileSystemLabel "Data" -Confirm:$false
```

Reclaim Host Storage (Trim/Discard) Issues TRIM/UNMAP commands to hypervisor storage pools to reclaim deleted blocks

```
$ Optimize-Volume -DriveLetter C -Defrag -Verbose
```

List Running Background Services Displays all active background services in a structured table

```
$ Get-Service | Where-Object Status -eq 'Running' | Format-Table Name, DisplayName, StartType
```

Restart Service Forces a controlled restart of an operational background service

```
$ Restart-Service -Name sshd -Force
```

Audit Top 10 Memory-Consuming Processes Sorts active processes descending by their physical memory working set

```
$ Get-Process | Sort-Object WorkingSet64 -Descending | Select-Object -First 10 -Property Id, ProcessName, CPU, @{Name='RAM_MB';Expression={[math]::Round($_.WorkingSet64/1MB,2)}} | Format-Table
```

Terminate Process by PID Stops an unresponsive or misbehaving process by its process identifier

```
$ Stop-Process -Id 4328 -Force
```

Audit Local User Accounts Lists all local user accounts along with enabled status and last logon timestamps

```
$ Get-LocalUser | Format-Table Name, Enabled, PasswordRequired, PasswordLastSet, LastLogon
```

Create Secondary Local Administrator Provisions a local account and appends it to the local Administrators security group

```
$ New-LocalUser -Name "secadmin" -Password $Password; Add-LocalGroupMember -Group "Administrators" -Member "secadmin"
```

List Installed Roles and Features Displays all installed Windows roles and operating system features

```
$ Get-WindowsFeature | Where-Object Installed -eq $true | Format-Table Name, DisplayName
```

Install Windows Role with Management Tools Installs a server role along with associated management cmdlets

```
$ Install-WindowsFeature -Name DNS -IncludeManagementTools
```

Query and Install Windows Updates Downloads all pending quality and security patches and triggers reboot if required

```
$ Install-WindowsUpdate -MicrosoftUpdate -AcceptAll -AutoReboot
```

Audit Hotfix and Patch History Lists installed updates ordered descending by installation timestamp

```
$ Get-HotFix | Sort-Object InstalledOn -Descending | Format-Table HotFixID, Description, InstalledOn
```

Update Microsoft Defender Signatures Fetches latest antivirus definitions online and applies them immediately

```
$ Update-MpSignature
```

Initiate Antivirus QuickScan Executes a rapid background scan of critical operating system paths

```
$ Start-MpScan -ScanType QuickScan
```

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

Configure Path Exclusion in Defender Prevents real-time scanning overhead on performance-critical database storage paths

```
$ Add-MpPreference -ExclusionPath "C:\Data\Database"
```

Audit VSS Shadow Copy Writers Validates that all registered shadow copy writers report state 1 Stable

```
$ vssadmin list writers
```

Ping QEMU Guest Agent from Host Pings the guest agent socket from the Proxmox host shell to confirm communication

```
$ qm guest cmd 104 ping
```

Audit NTP Time Synchronization Status Displays stratum, peer source, precision, and clock drift

```
$ w32tm /query /status
```

Force Immediate NTP Resynchronization Triggers immediate clock synchronization against configured NTP peers

```
$ w32tm /resync /rediscover
```

Install VirtIO Drivers Recursively from ISO Imports all paravirtualized drivers into the Windows Driver Store

```
$ pnputil.exe /add-driver D:\*.inf /subdirs /install
```

Verify Hardware and Driver Status Identifies hardware components whose status diverges from OK

```
$ Get-PnpDevice | Where-Object Status -ne 'OK'
```

Configure Switch-Independent NIC Team Binds multiple network adapters into a resilient team without switch dependencies

```
$ New-NetLbfoTeam -Name "Team01" -TeamMembers "Ethernet0", "Ethernet1" -TeamingMode SwitchIndependent -LoadBalancingAlgorithm Dynamic
```

Append System PATH Persistently Persists an administrative script path globally at the machine level

```
$ [System.Environment]::SetEnvironmentVariable("Path", "$([System.Environment]::GetEnvironmentVariable('Path','Machine'));C:\Scripts", "Machine")
```

Create Storage Pool and Mirrored Virtual Disk Aggregates unpartitioned disks into a fault-tolerant software mirror volume

```
$ New-StoragePool -FriendlyName "DataPool" -StorageSubsystemFriendlyName "Windows Storage*" -PhysicalDisks (Get-PhysicalDisk -CanPool $true); New-VirtualDisk -StoragePoolFriendlyName "DataPool" -FriendlyName "MirrorVolume" -ResiliencySettingName Mirror -UseMaximumSize
```

Install IIS Web Server and Management Tools Activates the Internet Information Services role and management cmdlets

```
$ Install-WindowsFeature -Name Web-Server, Web-Mgmt-Tools
```

Provision Isolated IIS Website and AppPool Configures an isolated web application pool and website bound to port 80

```
$ New-WebAppPool -Name "AppPool"; New-WebSite -Name "Site" -Port 80 -PhysicalPath "C:\inetpub\site" -ApplicationPool "AppPool"
```

Generate Self-Signed X.509 Certificate Creates a 2048-bit RSA test certificate in the local machine store

```
$ New-SelfSignedCertificate -DnsName "win-core01.lab.internal" -CertStoreLocation "Cert:\LocalMachine\My" -NotAfter (Get-Date).AddYears(2)
```

Import Enterprise CA Root Certificate Imports a trusted enterprise root certificate into the local machine store

```
$ Import-Certificate -FilePath "C:\Certs\InternalCA.crt" -CertStoreLocation "Cert:\LocalMachine\Root"
```

Apply NTFS Access Control Lists (ACLs) Writes explicit file system access control rules for users and groups

```
$ Set-Acl -Path "D:\Shares\Deploy" -AclObject $Acl
```

Active Directory Domain Join Joins an Active Directory domain and reboots the host immediately

```
$ Add-Computer -DomainName "corp.example.com" -Credential (Get-Credential) -Restart -Force
```

Create Primary DNS Forward-Lookup Zone Provisions a new authoritative forward-lookup zone on the local DNS server

```
$ Add-DnsServerPrimaryZone -Name "lab.internal" -ZoneFile "lab.internal.dns"
```

Add Static DNS A Record Registers a static IPv4 host record within the specified DNS zone

```
$ Add-DnsServerResourceRecordA -ZoneName "lab.internal" -Name "gateway" -IPv4Address "192.0.2.1"
```

Configure SMB Network Share Provisions an SMB network share granting Full Control to local Administrators

```
$ New-SmbShare -Name "Deploy" -Path "D:\Shares\Deploy" -FullAccess "BUILTIN\Administrators"
```

Register Weekly Maintenance Scheduled Task Schedules a weekly PowerShell maintenance script running with SYSTEM privileges

```
$ Register-ScheduledTask -TaskName "Maintenance" -Action (New-ScheduledTaskAction -Execute "powershell.exe" -Argument "-File C:\Scripts\Maintain-ServerCore.ps1") -Trigger (New-ScheduledTaskTrigger -Weekly -DaysOfWeek Sunday -At 03:00) -Principal (New-ScheduledTaskPrincipal -UserId "NT AUTHORITY\SYSTEM" -RunLevel Highest)
```

Query Live Performance Counters Reads real-time metrics for CPU utilization, free memory, and disk queues

```
$ Get-Counter -Counter "\Processor(_Total)\% Processor Time", "\Memory\Available MBytes"
```

Log Performance Counters to CSV Records performance counters over an interval and exports metrics to CSV

```
$ Get-Counter -Counter "\Processor(_Total)\% Processor Time" -SampleInterval 5 -MaxSamples 60 | Export-Counter -Path "C:\perf.csv" -FileFormat CSV
```

Disable LLMNR Multicast Resolution Hardens against local name spoofing and credential theft by disabling LLMNR

```
$ Set-ItemProperty -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows NT\DNSClient" -Name "EnableMulticast" -Value 0 -Type DWord
```

Disable NetBIOS over TCP/IP Deactivates legacy NetBIOS broadcasts across all IP-enabled network interfaces

```
$ Get-CimInstance Win32_NetworkAdapterConfiguration | Where-Object IPEnabled | Invoke-CimMethod -MethodName SetTcpipNetbios -Arguments @{TcpipNetbiosOptions=2}
```

Enforce NTLMv2 and Block NTLMv1 Configures LSA authentication level to strictly require NTLMv2 session security

```
$ Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\Lsa" -Name "LmCompatibilityLevel" -Value 5 -Type DWord
```

Clean WinSxS Component Store and Reset Base Permanently removes superseded update payloads, reclaiming up to 6 GiB of storage

```
$ dism.exe /Online /Cleanup-Image /StartComponentCleanup /ResetBase
```

Start Circular Network Packet Capture Captures raw network traffic into a circular Microsoft ETL trace buffer

```
$ netsh trace start capture=yes tracefile=C:\trace.etl maxsize=256 filemode=circular overwrite=yes
```

Stop Network Packet Capture Terminates active packet recording and finalizes the ETL trace file

```
$ netsh trace stop
```

Transfer File from Linux to Windows via SCP Copies scripts or binaries over encrypted SSH directly into Windows storage

```
$ scp -i ~/.ssh/id_ed25519 setup.ps1 Administrator@192.0.2.10:C:/Scripts/
```

Tunnel Internal Service via SSH Local Port Forwarding Tunnels an internal Windows service port securely to your local Linux workstation

```
$ ssh -i ~/.ssh/id_ed25519 -L 8080:localhost:80 Administrator@192.0.2.10
```

Standardize Locale and Keyboard Layout Enforces system language, regional format culture, and keyboard layout

```
$ Set-WinUserLanguageList (New-WinUserLanguageList "en-US") -Force; Set-WinSystemLocale "en-US"; Set-Culture "en-US"
```

Create Hypervisor Snapshot before Maintenance Freezes virtual machine hardware and disk state prior to applying major updates

```
$ qm snapshot 104 "pre-upgrade" --description "Snapshot prior to cumulative update cycle"
```

Roll Back Hypervisor Snapshot Reverts the virtual machine instantly to a saved hypervisor snapshot

```
$ qm rollback 104 "pre-upgrade"
```

Execute Application-Consistent vzdump Backup Triggers an application-consistent snapshot backup utilizing the QEMU Guest Agent

```
$ vzdump 104 --storage local --mode snapshot --compress zstd
```

Unlock Local User Account Clears account lockout following repeated failed authentication attempts

```
$ Unlock-LocalUser -Name "Administrator"
```

Expand VSS Shadow Storage Allocation Increases maximum disk capacity reserved for Volume Shadow Copies

```
$ vssadmin resize shadowstorage /for=C: /on=C: /maxsize=25%
```

Flush DNS Resolver Cache Purges stale DNS records from the local client resolver cache

```
$ Clear-DnsClientCache
```

Verify System File Integrity via SFC Scans and repairs corrupted protected operating system binaries

```
$ sfc.exe /scannow
```

Repair Component Store Online via DISM Repairs corrupted Windows image binaries directly using Windows Update

```
$ dism.exe /Online /Cleanup-Image /RestoreHealth
```

Query System Boot Timestamp and Uptime Retrieves exact timestamp of last operating system boot via CIM

```
$ Get-CimInstance Win32_OperatingSystem | Select-Object LastBootUpTime
```

Purge Recycle Bin on All Drives Clears recycled files permanently across all storage volumes without confirmation

```
$ Clear-RecycleBin -Force -Confirm:$false
```

List Active Listening TCP Sockets Displays local listening sockets along with owning process IDs

```
$ Get-NetTCPConnection -State Listen | Format-Table LocalAddress, LocalPort, OwningProcess
```

Create Custom Inbound Firewall Rule Opens an inbound firewall port for a custom application service

```
$ New-NetFirewallRule -Name "App-TCP-In" -DisplayName "App Service" -Direction Inbound -Protocol TCP -LocalPort 8080 -Action Allow
```

Purge Stale SSH Host Key on Linux Client Removes outdated host fingerprint from known_hosts following VM redeployment

```
$ ssh-keygen -R 192.0.2.10
```

Configure WinRM TrustedHosts for Standalone Servers Allows remote management connections to non-domain standalone hosts without server certificate validation

```
$ Set-Item WSMan:\localhost\Client\TrustedHosts -Value "192.0.2.10" -Force
```

Validate DNS Resolution to Windows Update Verifies name resolution for Microsoft update endpoints when debugging error 0x8024402C

$ `Resolve-DnsName update.microsoft.com`

Reset Windows Update Download Cache Purges corrupted update staging directories and restarts update agent service

$ `Stop-Service wuauserv; Remove-Item C:\Windows\SoftwareDistribution\Download\* -Recurse -Force; Start-Service wuauserv`

Bring Disk Online and Clear Read-Only Attribute Resolves offline or write-protected status on newly attached SAN LUNs or virtual disks

$ `Set-Disk -Number 1 -IsOffline $false; Set-Disk -Number 1 -IsReadOnly $false`

Restart VSS and QEMU Guest Agent Restarts hanging Volume Shadow Copy components during hypervisor backup stalls

$ `Restart-Service vss, QEMU-GA -Force`

Controlled System Reboot Initiates an immediate controlled reboot of the server via PowerShell

$ `Restart-Computer -Force`

Controlled System Shutdown Shuts down the operating system and powers down the virtual machine cleanly

$ `Stop-Computer -Force`

Audit BitLocker Encryption Status Displays encryption percentage, protection state, and key protectors across all volumes

$ `Get-BitLockerVolume | Format-Table MountPoint, VolumeStatus, EncryptionPercentage, KeyProtector`

Enable BitLocker on Operating System Drive Enables XTS-AES 256-bit encryption on volume C: utilizing the virtual TPM

$ `Enable-BitLocker -MountPoint "C:" -EncryptionMethod XtsAes256 -UsedSpaceOnly -TpmProtector`

Retrieve BitLocker 48-Digit Recovery Key Queries and extracts numerical disaster recovery password for offsite backup

```
$ (Get-BitLockerVolume -MountPoint "C:").KeyProtector | Where-Object KeyProtectorType -eq 'RecoveryPassword' | Select-Object -ExpandProperty RecoveryPassword
```

Create Reverse DNS Lookup Zone Provisions an IPv4 reverse lookup zone for subnet 192.0.2.0/24

```
$ Add-DnsServerPrimaryZone -NetworkId "192.0.2.0/24" -ZoneFile "2.0.192.in-addr.arpa.dns"
```

Add Reverse DNS PTR Host Record Registers an IP-to-hostname pointer within the authoritative reverse lookup zone

```
$ Add-DnsServerResourceRecordPtr -ZoneName "2.0.192.in-addr.arpa" -Name "10" -PtrDomainName "win-core01.lab.internal."
```

Add DNS Mail Exchanger Record Configures primary MX record with preference 10 for domain routing

```
$ Add-DnsServerResourceRecordMX -ZoneName "lab.internal" -Name "@" -MailExchange "mail.lab.internal." -Preference 10
```

Discover Active Directory Domain Controllers Identifies active directory domain controllers and forest topology via PowerShell

```
$ Get-ADDomainController -Discover
```

Audit Active Domain User Accounts Queries enabled domain user accounts within specified organizational units

```
$ Get-ADUser -Filter 'Enabled -eq $true' | Format-Table SamAccountName, Name, UserPrincipalName
```

Export Windows Event Logs to EVTX File Exports system and security event channels to offline EVTX files for archival

```
$ wevtutil.exe epl System "C:\Scripts\LogArchive\System.evtx"
```

Compress Event Log Archives to ZIP Compresses exported log files into consolidated archive and purges raw EVTX artifacts

```
$ Compress-Archive -Path "C:\Scripts\LogArchive\*.evtx" -DestinationPath "C:\Scripts\LogArchive\Logs.zip" -Force
```

Enforce Script Execution Policy Configures machine-level PowerShell execution policy to RemoteSigned

```
$ Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope LocalMachine -Force
```

Unblock Transferred PowerShell Scripts Removes alternate data streams from scripts copied from Linux or downloaded from the web

```
$ Unblock-File -Path "C:\Scripts\*.ps1"
```

Run System Health Verification Audit Executes automated health audit script inspecting memory, disk, services, and VSS writers

```
$ powershell.exe -NoProfile -ExecutionPolicy Bypass -File C:\Scripts\Test-ServerHealth.ps1
```

Out-of-Band Emergency Recovery via Proxmox noVNC Console

If a misconfigured firewall rule, damaged network configuration, or syntax error in `sshd_config` severs remote network access, the hypervisor console serves as your primary lifeline.

Execute this structured recovery procedure:

1. **Open noVNC Console in Proxmox Dashboard:**

Navigate to virtual machine `104` and launch the **Console** (noVNC).

1. **Trigger Logon Screen:**

Click the **`Send Ctrl-Alt-Del`** action in the top-left noVNC menu toolbar.

1. **Log On Locally:**

Authenticate using the local `Administrator` account.

1. **Spawn Emergency Shell:**

If the screen is blank, press `Ctrl` + `Shift` + `Esc` to launch Task Manager. Navigate to *File* → *Run new task*, enter `powershell.exe`, and check the box *„Create this task with administrative privileges“*. **5. Reset Firewall and Network Bindings:**

Reset the firewall to factory defaults to eliminate blocking rules:

```
netsh advfirewall reset
```

Launch SConfig to return network adapters to DHCP or assign verified static parameters:

```
sconfig
```

6. Audit and Restart OpenSSH Service:

Confirm daemon operational state within PowerShell:

```
Restart-Service sshd
Get-NetTCPConnection -LocalPort 22 -State Listen
```

With port 22 listening and firewall rules reset, terminal access from your Linux workstation is restored.

Further Resources

The following official documentation, standards, and AdminDocs modules provide deeper technical exploration:

Microsoft Learn: What's New in Windows Server 2025 Official overview of platform architecture, security enhancements, and server roles in Windows Server 2025

Microsoft Learn: Manage Server Core Official documentation regarding management options, supported roles, and maintenance on Server Core

Microsoft Learn: SConfig Configuration Reference Complete operational reference and parameter documentation for the Server Configuration tool

Microsoft Learn: OpenSSH for Windows Installation, configuration, and security guide for the native Windows OpenSSH daemon

Microsoft Learn: PowerShell Architecture and Scripting Official documentation covering PowerShell pipelines, WMI/CIM management, and module mechanics

Proxmox VE Administration Guide: QEMU/KVM Official hypervisor manual detailing virtual machine architecture, controllers, vTPM, and hardware tuning

Proxmox Wiki: Windows Guest Best Practices Official guidance on configuring Windows Server 2025 with VirtIO drivers and QEMU Guest Agent

Proxmox VE: Installation and First VM Comprehensive guide covering hypervisor setup, storage pool attachment, and network bridges

Linux Administration: Virtualization and VM Management In-depth exploration of KVM, QEMU, and VirtIO paravirtualization architectures

Linux Administration: System Security and Hardening Datacenter security engineering, network firewall segmentation, and SSH daemon hardening

Conclusion

Operational Decision Matrix and Architectural Evaluation

Deploying Windows Server 2025 Core is a foundational architectural commitment that extends far beyond provisioning a single virtual machine. For datacenter infrastructure planning, headless server operations present distinct technical criteria defining where Server Core is mandatory—and where the traditional Desktop Experience remains unavoidable.

When Server Core is the Definitive Architecture:

- **Core Infrastructure Roles:** Active Directory Domain Services (AD DS), DNS, DHCP, Active Directory Certificate Services (AD CS), and File and Storage Services operate on Server Core with unmatched uptime, minimal memory overhead, and freedom from disruptive desktop update reboots.
- **Heterogeneous Cross-Platform Fleets:** With native OpenSSH Server integrated directly into Windows Server 2025, Windows hosts integrate seamlessly into Linux-driven automation frameworks (Ansible, Terraform, CI/CD runners) without exposing proprietary RPC ports or fragile WinRM HTTP/HTTPS certificate chains across firewall boundaries.
- **Dense Virtualization Clusters:** On hypervisors such as Proxmox VE 9, the tiny memory baseline (less than 1.5 GiB idle RAM) significantly increases VM consolidation ratios per host node while accelerating hypervisor snapshot and backup windows.

Operational Exclusion Criteria (When Desktop Experience is Unavoidable):

- **Interactive End-User Desktop Sessions:** Server Core cannot function as a Remote Desktop Session Host (RDSH / Terminal Server) for end users.
- **Third-Party Legacy Software Dependencies:** Certain commercial enterprise applications, legacy database engines, or proprietary backup agents require Windows GDI and Explorer desktop libraries to execute installer wizards or licensing validation routines.
- **Specialized Management Roles:** Specific roles (such as Fax Server) or legacy software configured exclusively through local MMC snap-ins without remote RSAT support continue to require the full desktop environment.

Operational Competence and Disaster Recovery:

Transitioning to Server Core shifts administrative responsibility from hypervisor compute capacity to engineering discipline: without structured PowerShell proficiency and an established command reference, incident triage stalls.

If an overly restrictive firewall rule or misconfigured network adapter severs SSH reachability, Server Core provides no graphical fallback. The primary recovery mechanism in such scenarios is out-of-band hypervisor access via Proxmox noVNC: authenticate at the console, restore baseline networking via `sconfig`, or remediate adapter configurations directly within local PowerShell.

Source: https://admindocs.de/en/windows-server/windows-server-2025-core-installation-and-administration

When evaluated within these operational boundaries, Windows Server 2025 Core on Proxmox VE 9 establishes a remarkably resilient, low-maintenance, and fully automatable pillar for modern enterprise infrastructure.